LIMBAJUL FoxPro

NOȚIUNEA DE ALGORITM

Această lucrare o fost elaborată pentru a veni în sprijinul celor care doresc să învețe şi să practice programarea în limbajul PASCAL. Lucrarea trebuie privită ca o introducere în problematica limbajului PASCAL. Odată stăpânite noțiunile prezentate în continuare, oricine poate aprofunda oricât şi în orice direcție toate subtilitățile diverselor variante ale limbajului TURBOPASCAL.

Scrierea algoritmului de rezolvare a problemei.

Definiție: - algoritmul este un ansamblu de reguli de prelucrare, împreună cu ordinea în care se succed în vederea soluționării unui tip de probleme.

- -un sistem de reguli care pentru o problemă dată, pornind de la datele inițiale se ajunge la rezultatele finale pe baza unui proces unic, finit, proces deschis printr-o succesiune de operații de rutină la care nu intervine aportul creator al omului.

Proprietățile algoritmului

- generalitatea – constă în aceea că un algoritm nu rezolvă o singură problemă ci o clasă de probleme de acelaşi tip;
- Finititudinea – numărul transformărilor ce trebuie aplicat unei informații de intrare pentru a obține imformația finală este finit;

Unicitatea – toate transformările prin care trece informația finală sunt univoc determinate de regulile algoritmului.

Mărimi cu care operează algoritmii

Variabile - o mărime care poate lua o mulțime de valori posibile în cursul prelucrării.

Constante - o mărime ce are atribuită o valoare care nu se modifică în timpul execuției.

Mărimile pot fi succesiuni de caractere alfabetice, numerice şi chiar speciale. Este indicat ca aceste numere atribuite mărimilor să fie sugestive.

Operații utilizate în algoritmi

Într-un algoritm, regulile trebuie să precizeze foarte clar operațiile ce se execută asupra datelor.

1.Operații de calcul – sunt operațiile obişnuite de : adunare (+), scădere (-), înmulțire (*), împărțire (/), ridicare la putere.

Acestea intervin în cadrul expresiilor care sunt o succesiune de variabile şi constante legate între ele prin operatori (semne de operații) şi eventual paranteze, după reguli bine definite.

În cadrul expresiilor operațiile se execută în ordinea naturală, conform priorităților.

Într-un algoritm o expresie apare întotdeauna în cadrul unei operații de atribuire.

2.Operații de atribuire: printr-o asemenea operație se atribuie unei variabilie o valoare a unei - constante, variabile, expresii.

Operația de atribuire se notează cu: „ := „ sau „←"

Ex: NUME : = 'IOAN' (constantă) NUME ← 'IOAN'

NUME : = NUMEP (variabilă) NUME ← NUMEP

A:= 1, A←1, A:= X-1, A:= A+1

3.Operații de test (decizie): scopul acestei operații este de a verifica relațiile existente între datele asupra cărora operează algoritmul pentru a decide transmiterea controlului execuției către o anumită instrucțiune.

În urma executării unei operații de test rezultatul obținut este una din aşa numitele valori logice de adevăr: „ adevărat" sau „fals".

Operațiile de test se reprezintă prin semnele: <;

4.Operații de intrare/ieşire : se referă la introducerea datelor de intrare respectiv furnizarea rezultatelor.

Operații de intrare – citire, atribuire – citeşte

Operații de ieşire - scriere, afişare –scriere

Simboluri grafice

Schema logică este forma grafică de reprezentare a unui algoritm utilizând simbolurile de reprezentare a operațiilor.

Simbol	Denumire simbol	Semnificație
	Săgeată	Realizează legătura între blocuri
	Bloc delimitator	Marchează începutul sau sfârşitul programului
	Bloc de intrare/ieşire	Reprezintă operația de citire sau scriere
	Bloc de calcul sau atribuire	Descrie operațiile de calcul şi atribuire
DA NU	Bloc de decizie	Reprezintă condițiile puse
a) b)	Conectori	a)Când mai multe variante ale schemei logice se întâlnesc în acelaşi punct, conector de pagină b) conector de trecere pe altă pagină
	Bloc de procedură	Folosit pentru apelarea unei proceduri (program) reprezentat anterior

Operații

- - aritmetice: avem operații cu - numere întregi (+, - , *, DIV, MOD)
- - numere reale (+, -, *, /)

- operații pe biți – se aplică numai operanzilor de tip întreg. Operatori pot fi:

AND SI aritmetic (conjuncție)
OR SAU aritmetic (disjuncție)
XOR SAU EXCLUSIV
NOT NEGAȚIE (înlocuirea lui 0 cu 1 şi invers)
SHL deplasare stânga bit cu bit
SHR deplasare dreapta bit cu bit

Deplasare se face pe toată lungimea în biți a tipului la care se aplică.

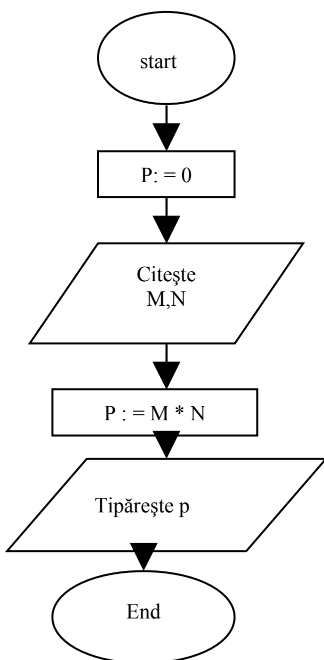
Operație	Rezultat
0 OR 0	0
0 OR 1	1
1 OR 0	1
1 OR 1	1
0 AND 0	0
0 AND 1	0
1 AND 0	0
1 AND 1	1
NOT 0	1
NOT 1	0

- operații cu mulțimi – reuniune, intersecția, diferența.

- - operații cu şiruri de caractere – operanzii pot fi de tip şir de caractere (string) sau de tip chart. Rezultatul este întotdeauna de tip şir de caractere
- - operația de adresare – se aplică asupra identificatorilor de variabilă, constantă simbolică, funcții, proceduri
- - operații logice (AND, OR, XOR, NOT) – operanzii sunt de tip logic (Boolean).
- - operații relaționale (relații)- permit compararea a doi operanzi, rezultatul va fi de tip boolean.

Exemple de scheme logice

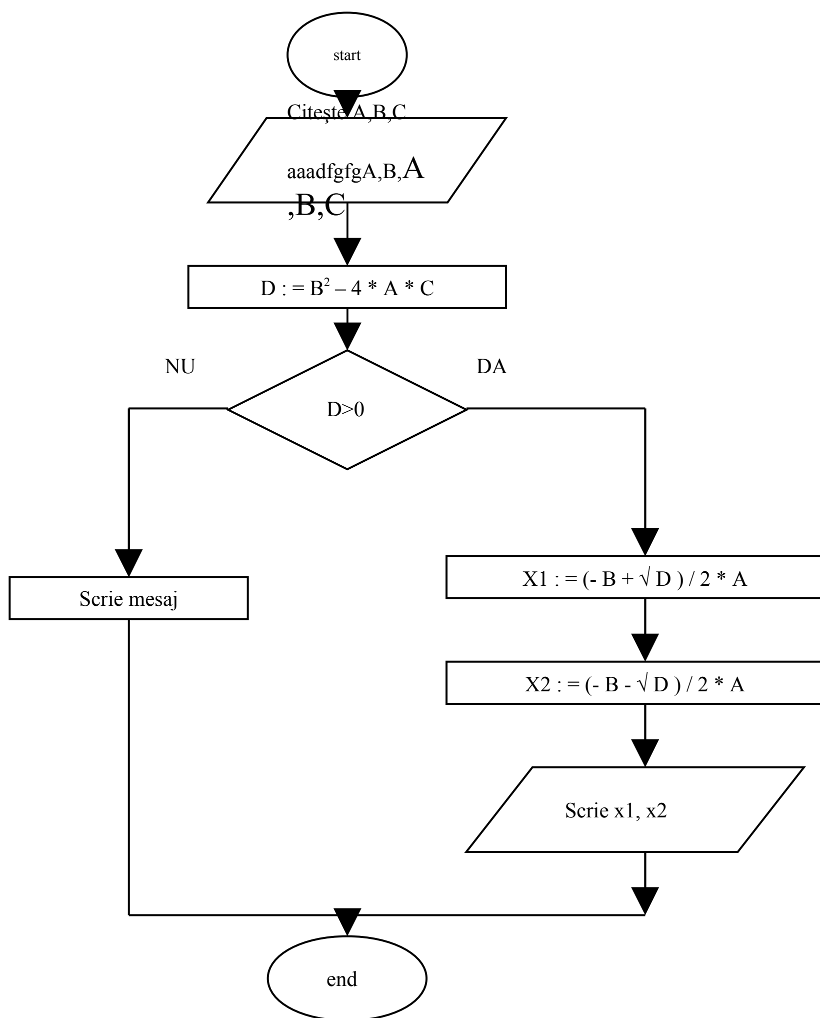
1. Schema logică a înmulțirii



2. Schema logică a algoritmului care rezolvă ecuația de gradul 2 de forma:

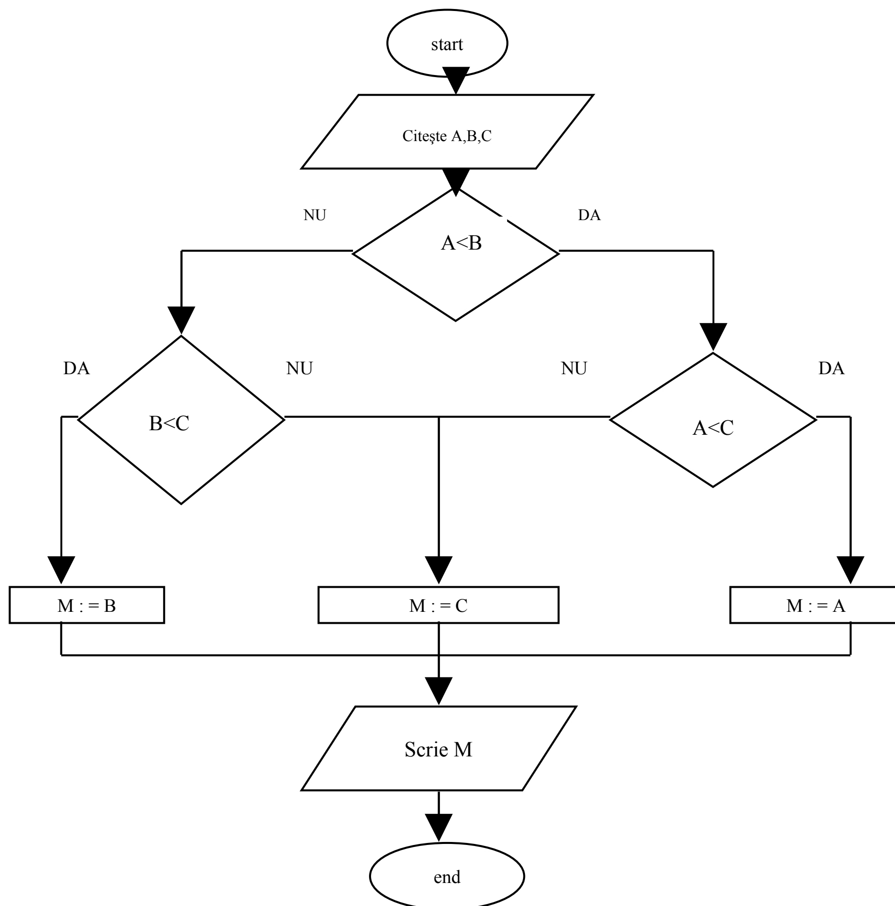
$Ax^2 + Bx + C = 0$

Pentru rezolvare se calculează $D := B^2 - 4 * A * C$, iar apoi valorile pentru x1 şi x2 în cazul când acestea au valori reale, iar în cazul când x1 şi x2 au valori complexe se scrie un mesaj.



3. Să se deseneze schema logică a unui algoritm de calcul al valorii minime din trei numere date (A, B, C) şi şă se tipărescă această valoare.

Pentru a găsi minimul a trei elemente se compară primele două şi cel mai mic dintre ele se compară cu al treilea. Cel mai mic dintre acestea este elementul cu valoarea minimă.

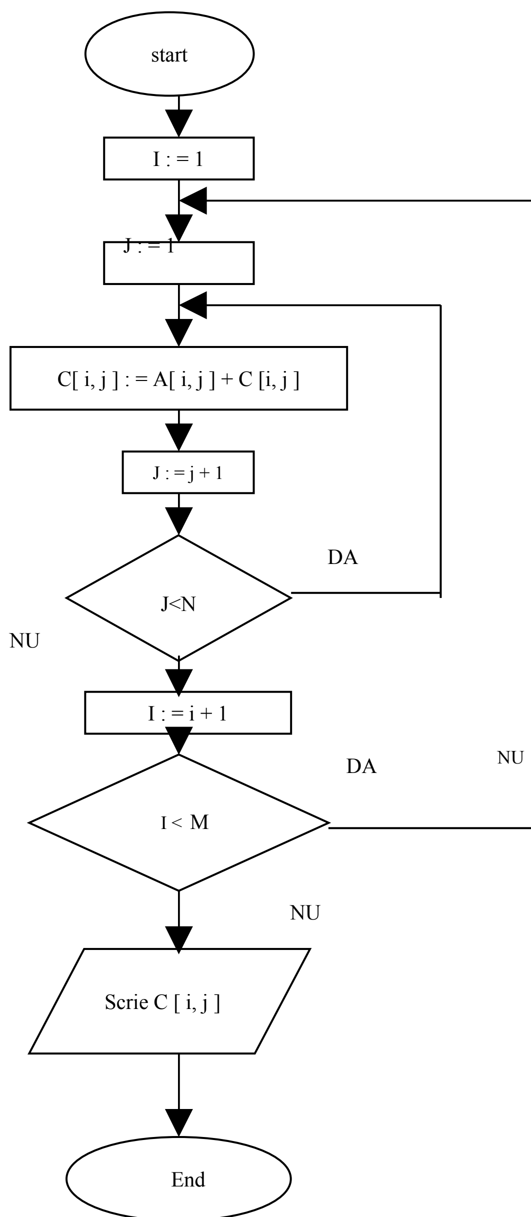


4. Să se deseneze schema logică pentru suma a două matrici.

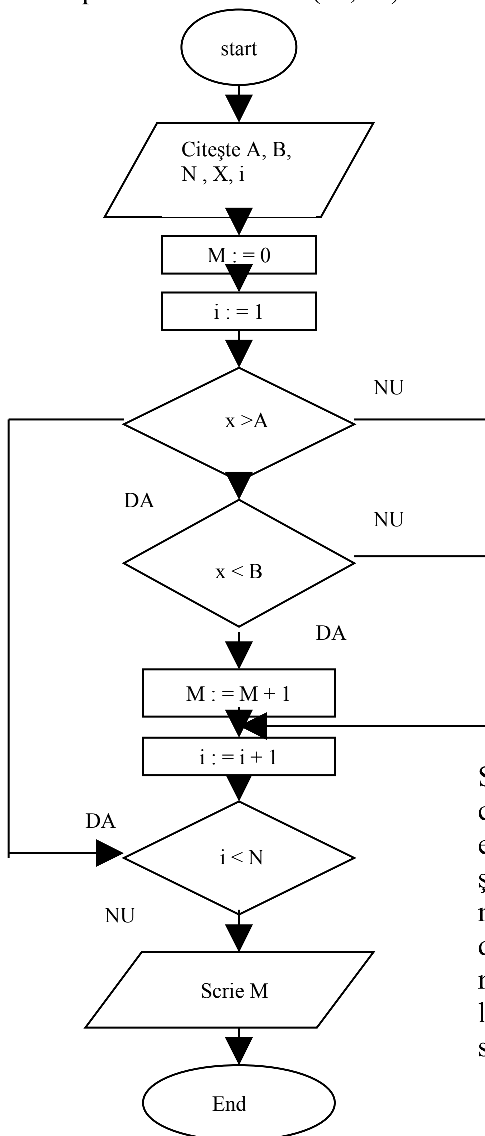
Operaţiile asupra matricilor impun parcurgerea acestora element cu element, fapt care presupune modificarea ciclică a indicilor de linie şi de coloană. Presupunem că avem matricea A [M, N] şi matricea B [M, N] iar matricea rezultat va fi C [M, N] unde M reprezimtă numărul de linii iar N reprezintă numărul de coloane.

I = 1, M

J = 1, N

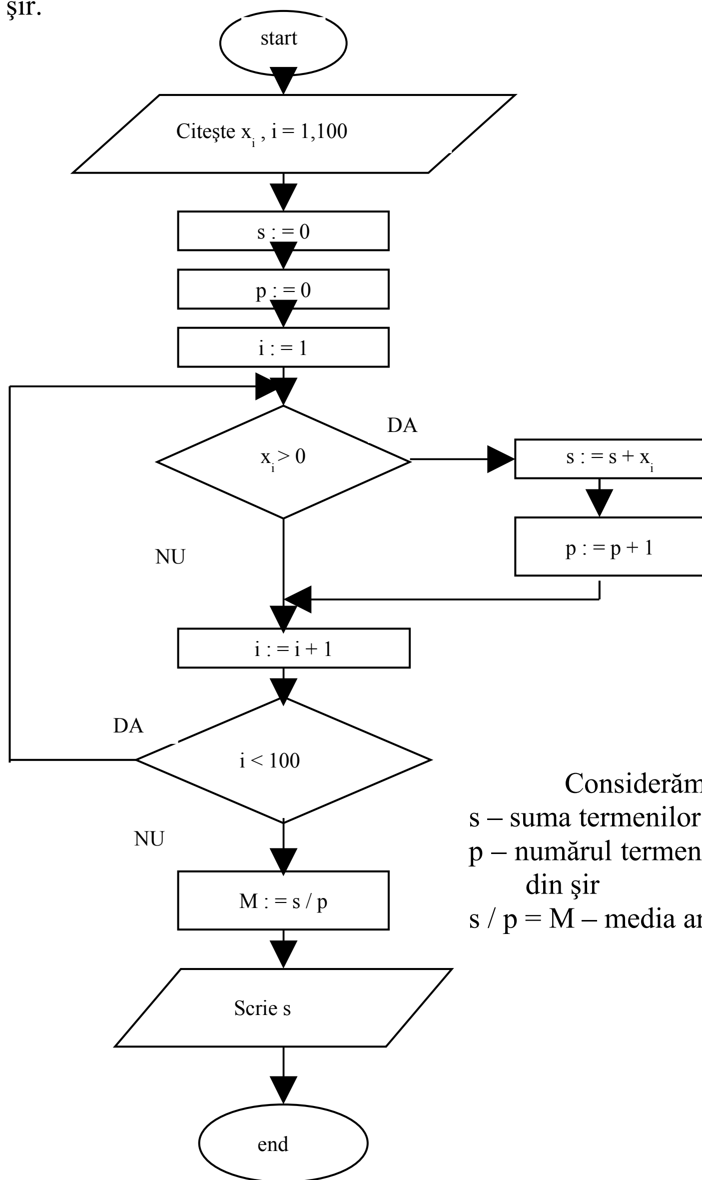


5. Se consideră un şir de N numere reale x1, x2, ...xn şi numerele reale A şi B. Să se deseneze schema logică al unui algoritm de calcul al numărului de elemente din şir care sunt cuprinse în intervalul (A, B).



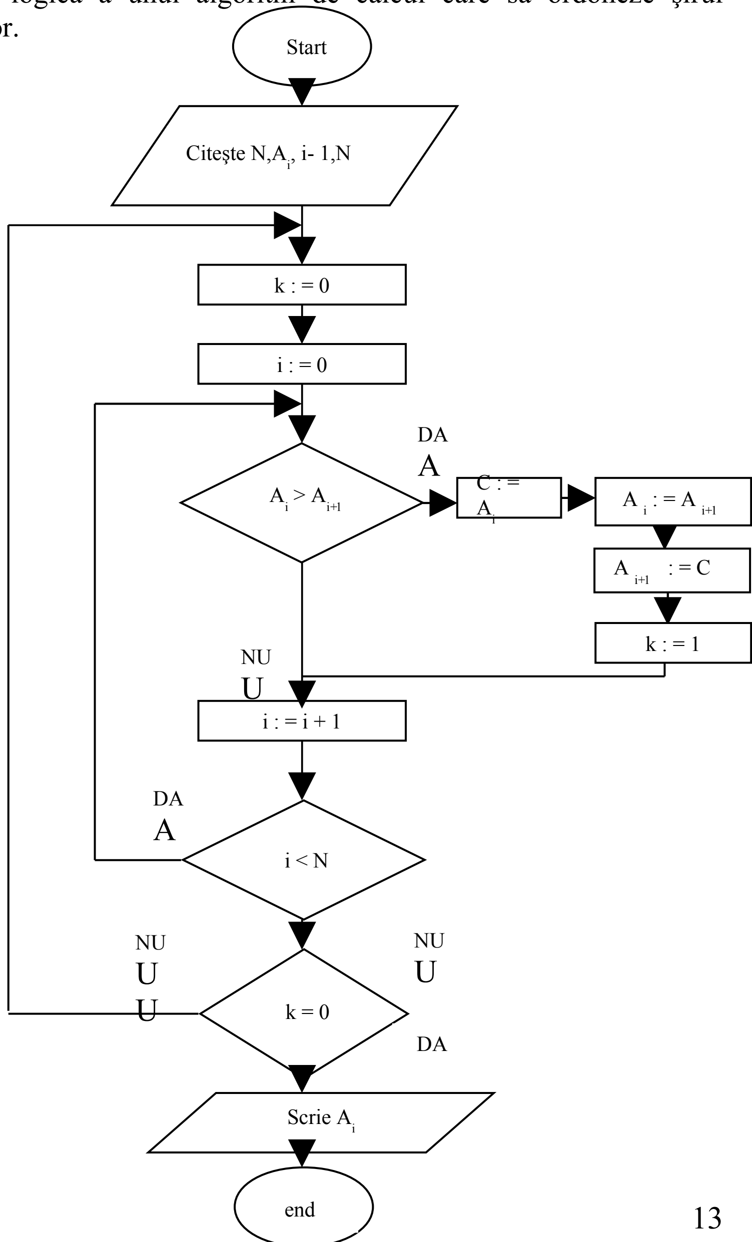
Se iniţializează numărul M cu 0. se iau apoi elementele şirului pe rând şi dacă sunt în interval se măreşte contorul M cu 1, dacă nu M rămâne neschimbat. Apoi se trece la următorul element s.a.m.d.

6.Se consideră şirul $x_1 \ldots x_{100}$. Să se deseneze schema logică a unui algoritm de calcul a mediei aritmetice a termenilor pozitivi din şir.



Considerăm:
s – suma termenilor pozitivi
p – numărul termenilor pozitivi din şir
s / p = M – media aritmetică

7.Se dă un şir de N numere reale $A_1 .. A_n$. Să se deseneze schema logică a unui algoritm de calcul care să ordoneze şirul crescător.



Etapele de execuție a unui program

Limbajul de programare constituie mijlocul de comunicare între om şi calculator, iar descrierea succesiunii de operații pe care trebuie să le efectueze calculatorul se numeşte program.

Etapele necesare realizării unui program sunt următoarele:

a) scrierea programului (într-un anumit limbaj de programare);

b) compilarea programului;

c) editarea de legături (link-editarea);

d) execuția programului;

e) eventuala depanare a sa, reluând procedeul de la etapa a);

f) îmbunătățirea performanțelor programului;

a) Scrierea programului presupune editarea unui fişier ce conține textul programului. Acest fişier se cheamă fişier-sursă şi este un fişier text în format ASCII. În interiorul său el conține instrucțiuni ale limbajului în care se face programarea.

Un limbaj de programare este foarte asemănător limbajului obişnuit (natural); el reprezintă un sistem de convenții pe baza căruia se efectuează o comunicare. Deci limbajul de programare este un sistem de cuvinte (cheie), semne, construcții cu cuvinte şi semne care ne asigură de faptul că transmitem calculatorului ceea ce dorim să realizeze.

b)“Propozițiile” şi “frazele” unui limbaj de programare nu sunt direct înțelese de calculator. Ele sunt traduse din această formă uşor manevrabilă în instrucțiuni maşină (singurele recunoscute de microprocesor) de câte un program ce se cheamă compilator. Acest program recunoaşte noțiunile din fişierul-sursă şi le converteşte în echivalențele lor în limbaj maşină pe care le depune într-un fişier de ieşire numit fişier-obiect.

Fişierul-obiect produs de un compilator nu este, încă, gata de a fi executat necesitând o prelucrare suplimentară .

Dacă fiecare instrucţiune din fişierul-sursă ar produce instrucţiuni care realizează scrierea unui mesaj această porţiune de cod s-ar găsi repetată (cu mici modificări) în mai multe locuri din program. Pentru înlăturarea unui astfel de lucru nedorit, operaţiile cel mai des folosite sunt "izolate" într-o colecţie de rutine, de unde pot fi apelate oricând este nevoie.

Deci după compilare, fişierul-obiect produs nu va conţine întotdeauna instrucţiunile necesare unei operaţii ci eventuale referiri la rutina de bibliotecă ce execută operaţia în cauză.

c) Editarea de legături (link-editare) va rezolva aceste referiri stabilind conexiunile dintre referiri şi punctele referite. Ea va conecta referinţele nerezolvate de compilator cu modulele de bibliotecă ce le corespund.

Din colecţia de rutine (denumită bibliotecă) vor fi extrase numai acele rutine (module) referite care împreună cu instrucţiunile din fişierul-obiect vor constitui un program coerent, adică un fişier-executabil.

d) Execuţia programului este scopul final al etapelor anterioare şi presupune lansarea fişierului-executabil astfel obţinut. Pentru această operaţie sunt eventual necesare:

- îndeplinirea unor cerinţe din partea sistemului;
- specificarea de parametri corespunzători pentru program.

e) În cazul unui program mai amplu sau în cazul unui debut în programare va fi întotdeauna necesară depanarea programului obţinut deoarece (fără greş) acesta nu va funcţiona din prima. Pentru depistarea eventualelor erori stau la dispoziţie pachete de programe numite depanatoare.

f) După ce programul a fost convins să funcţioneze corect, abia atunci este cazul a încerca îmbunătăţirea performanţelor sale (viteză de execuţie, resurse sistem mai reduse, protejare la erori). Pentru acest reglaj "fin" stau la dispoziţie programe de tip "profiler" care detectează zonele de programe ce consumă cea mai mare cantitate de timp – puncte în care trebuie să se concentreze atenţia programatorului.

De regulă se obişnuieşte a se lansa o versiune de test a aplicaţiei care este exploatată efectiv pentru a constata eventualele erori de funcţionare.

Etapele de compilare şi link-editare sunt necesare pentru realizarea unui program compilat. Se pot “executa” aplicații şi fără producerea fişierelor obiect şi executabil. Metoda folosită este cea a unui interpretor.

Un interpretor, compilează, link-editează şi execută un program “linie-cu-linie”. Pe măsură ce sunt citite linii din fişierul-sursă ele sunt transformate în instrucțiuni maşină şi executate imediat. Pentru un program dat, un interpretor nu mai produce fişiere-obiect şi executabil, operând numai cu fişierul-sursă.

Care din metode este cea mai bună, cea a unui compilator sau cea a unui interpretor?

Pentru interpretor pledează un singur avantaj, şi anume reducerea timpului de punere la punct a unei aplicații (prin “eliminarea” etapelor intermediare), dar acest avantaj este infim, deoarece în momentul de față, mediile de programare de tip compilator, sunt extrem de rapide (sute de linii de text-sursă pe secundă) şi realizează automat link-editarea şi execuția, astfel încât întârzierile provocate de etapele “suplimentare” sunt neglijabile.

În schimb, avantajele unui program compilat față de un program interpretat sunt zdrobitoare:

- viteză de execuție de câteva ori (chiar zeci de ori) mai mare;
- posibilitatea de a rula de sine stătător; odată compilat, compilatorul nu este necesar în execuția programului, pe când un program interpretat nu se poate “executa” decât dacă interpretorul se află în memoria calculatorului;
- deci programul compilat dispune de resursele sistemului în întregime, pe când cel interpretat le împarte cu interpretorul (mai ales memoria, care este de multe ori critică);
- flexibilitatea sporită în realizarea programelor compilate (adăugarea de noi biblioteci, redefinirea unei rutine deja existente etc).

Toate aceste considerații au determinat ca interpretoarele să fie pe cale de dispariție la ora de față sau oricum, să fie dublate de un compilator. Spre exemplu limbajul BASIC ce beneficia inițial numai

de interpretor posedă în prezent şi compilatoare care încearcă să-l menţină în atenţia programatorilor.

Mediul de programare

Un pachet de programe ce asigură toate operaţiile de mai înainte poartă numele de mediu de programare.

Cu ajutorul său:

- se editează un program;
- se compilează şi eventual link-editează automat;
- se lansează în execuţie;
- se depanează un program;
- se poate "regla" execuţia unui program (cu un *profiler*).

Mediile de programare disponibile în prezent sunt deosebit de performante şi oferă utilizatorului toată gama de servicii necesare (help, exemple gata-construite etc), prin intermediul unor prgrame puternic interactive. Ele transformă munca laborioasă a programatorului într-un succes aproape sigur.

LIMBAJUL FOXPRO

Tipuri de fişiere

Fişierele FOXPRO reţin date, programe, informaţii pentru generatoarele sistemului şi se deosebesc prin extensii. Principalele fişiere cu care lucrează FoxPro sunt:

1. fişiere .DBF sunt fişierele baze de date;
2. fişiere .PRG sunt textele sursă ale programelor sau subprogramelor;
3. fişiere .MEM conţin variabilele de memorie;
4. fişierele .NDX conţin indexii asociaţi unei baze de date în vederea parcurgerii ordonate a acesteia;
5. fişierele .MDX conţin liste mai mari de indexi;
6. fişierele .SCR, .FMT sunt folosite de generatorul de machete ecran;
7. fişierele .FRM conţin proiectul de raport;
8. fişierele .LBL conţin proiectul de etichetă;

9. fişierele .DBO, .FRO, .FMO, .LBO sunt rezultatul compilării programelor şi proiectelor.

Operații generale cu fişiere şi directoare

Schimbarea directorului de lucru:

SET DIRECTORY TO <director>

Observație: putem folosi şi comanda DOS pentru această operație CD (change directory) apelând-o pentru execuție prin !.

Schimbarea discului curent :

SET DEFAULT TO <disc>

Observație: În FoxPro nu există comanda SET DIRECTORY; pentru poziționarea pe un anumit director putem folosi comanda SET DEFAULT TO <director>.
Indicarea directoarelor de căutare:

SET PATH TO <lista de directoare>

Orientarea căutării unui fişier într-o listă de directoare se face prin comanda SET PATH. Atenție: se va folosi <lista de directoare> la deschiderea unui fişier pentru consultare nu şi de la depunerea lui prin creare.

Variabile şi masive

Variabile

Prelucrarea datelor cu ajutorul calculatorului presupune mai întâi depozitare acestora într-o memorie externă sau interna a calculatorului. Pentru depozitarea în memoria internă a unei anumite date (de tip numeric, logic, şir de caractere etc.) se foloseşte variabila de memorie, sau simplu variabila. Aceasta reprezintă o zonă de memorie, căreia I se atribuie un nume, şi în care se poate memora o variabilă de un anumit tip. Ca urmare o variabilă are trei elemente specifice:

- numele variabilei, atribuit de programator sau predefinit de proiectanții FoxPro –ului, folosit pentr identificarea variabilei respective printre celelalte variabile existente la un moment dat în memorie;
- conținutul sau valoarea variabilei, reprezentând data ce este memorată în zona de memorie a veriabilei. În timpul unui program conținutul unei variabile se poate modifica;
- tipul variabilei, reprezentând tipul datei ce se poate memora în zona de memorie a variabilei. Acest tip determină comenzile şi functiile ce se pot aplica acestor variabile.

Exemplu:

Nume	Conținut + tip
alfa	432

Referirea la o variabilă se face prin numele său, astfel:

? alfa
432

se traduce prin “ afişează (?) conținutul variabilei alfa (432) “. variabila alfa este de tip numeric, înțelegând prin aceasta că ea poate memora un număr, o valoare de tip numeric.

Pentru folosirea unei variabile, ea trebuie mai întâi creată, acesata presupunțnd:

- atribuirea unui nume variabilei respective
- stabilirea tipului variabilei şi rezervarea zonei de memorie ce este atribuită variabilei;
- atribuirea unei valori inițiale pentru aceasta, adică stabilirea conținutului inițial al variabilei.

Tipurile posibile pentru variabile sunt: numeric, caracter, dată calendaristică, logic.

O altă caracteristică a variabilelor FOX este statutul lor public sau privat. O variabilă are statut privat (sau local) programului unde ea a fost creată, pierzându-şi valabilitatea după terminarea execuției programului. Dacă a fost creată în modul de lucru comandă ea are statutul de variabilă publică (globală) fiind recunoscută în toate unitățile funcționale (subprograme, funcții utilizator) lansate din acest mod de lucru.

Tablourile pot avea doar 1-2 dimensiuni şi – ceea ce este de subliniat – spre deosebire de limbajele de programare PASCAL, C, unde le-am mai întâlnit, în dBASE (FoxPro) *natura elementelor nu este omogenă*; în funcție de valoarea reținută la un moment dat, unele elemente pot fi numerice, altele caracter, etc.

Declararea variabilelor:

Variabilele simple nu necesită declarare, ci numai tablourile. Înainte de folosirea elementelor sale, tabloul trebuie declarat printr-o comandă:

DECLARARE <lista-tablou>

unde <lista-tablou> este o secvență de nume de tablouri separate de virgulă:
<nume-tab>[<dim1>[, <dim2>]].
Declararea dimensiunii unui tablou înseamnă şi inițializarea valorilor tabloului cu valoarea logică .F.
Observație: În FoxPro, pentru tablouri se pot folosi atât parantezele pătrate cât şi cele rotunde

Operația de atribuire:

În FoxPro, există două modalități de a atribui valori inițiale sau de a modifica valoarea unei variabile: prin comanda STORE şi prin comanda de atribuire <var>=<exp>.

a) Crearea unei variabile sau modificarea valorii acesteia se realizează prin operatorul de atribuire, cu următoarea sintaxă:

<variabilă> = <exp>

Funcționarea unei asemanea comenzi are loc astfel:

- se evaluează expresia <exp>, obținându-se o valoare de un anumit tip;
- se caută în memorie variabila cu numele <variabilă> şi, dacă se găseşte, se înlocuieşte vechiul conținut al acesteia, cu valoarea expresiei;
- dacă nu se găseşte variabila respectivă, FoxPro crează una nouă cu numele <variabilă>, în care depune valoarea expresiei;
- tipul variabilei este dat de tipul valorii expresiei, indiferent de tipul anterior al variabilei, în cazul când aceasta există şi înainte de execuția comenzii.

Exemplu:

a = 2 && se crează variabila numerică "a" cu valoarea inițială 2

a = "bună" && vechea variabilă "a" este înlocuită cu una nouă, de tip şir de caractere; ce va conține şirul "bună"

data = {12 \01 \70 } && se crează variabila "data", de tip calendaristic, cu valoarea inițială {12 \01 \70}

b) o comandă echivalentă cu operatorul de atribuire este comanda STORE:

STORE <expr> TO <listă-variabile>

Comanda constă în evaluarea expresiei <expr> şi depunerea valorii rezultate în toate variabilele din <listă variabile>. Cele care nu există vor fi create odată cu execuția comenzii.

Exemplu:

STORE 0 TO a, b, c

NOTE se crează trei variabile numerice a, b, c, care sunt inițializate cu valoarea 0.

STORE {12 \01 \70} TO data

NOTE se crează variabila data, de tip dată calendaristică şi se inițializează cu valoarea {12 \01 \70}

Citirea unei variabile

Operația de citire a unei variabile are înțelesul de atribuire unei valori de la tastatură pentru variabila respectivă. Sunt trei astfel de comenzi prin care se pot da valori unei variabile.
Atenție! numai unei singure variabile i se poate citi valoarea într-o comandă.

a) Comanda de citire INPUT permite crearea/modificarea oricărui tip de variabilă:

INPUT [<mesaj>] TO <var>

Comanda INPUT permite afişarea (eventuală) a unui mesaj <mesaj> pe ecran şi aşteaptă introducerea de către operator a unei expresii. Expresia se evaluează şi, dacă este corectă, se creează variabila cu numele specificat în <var>.
Variabile primeşte cu această ocazie tipul expresiei introdusă de operator.

b) comanda **ACCEPT** permite crearea sau modificarea variabilelor de tip caracter:

ACCEPT [<mesaj>] TO <var>

Observație: Indiferent de tipul expresiei introdusă de la tastatură, comanda consideră şi evaluează numai şiruri.

c) comanda WAIT permite crearea unei variabile de tip caracter şi lungime 1:

WAIT [<mesaj>] [TO <var>]

Comanda permite o pauză în program până când operatorul apasă o tastă.

Caracterul corespunzător tastei este eventual atribuit variabilei <var> imediat ce a fost apăsată tasta ne mai aşteptându-se certificarea introducerii prin tasta <enter>, ca la celelalte citiri. Şi mesajul poate lipsi; în acest caz sistemul are un mesaj standard de tipul:

„Press and key to continue”

Ştergerea variabilelor

Numărul de variabile pe care le poate gestiona sistemul este destul de mare ca să nu ne preocupe eliberarea spaţiului prin ştergerea unor variabile. De asemenea, toate variabilele folosite într-un program sunt şterse automat când programul respectiv se termină.

Dar, uneori este necesar să avem în memorie numai anumite variabile din cele manipulante în sesiune (pentru a le salva pe disc de exemplu).

Operaţia de ştergere a variabilelor are următorul format general:

RELEASE <listă-variabile>/ALL/ [LIKE /EXCEPT <macchetă>]

Comanda RELEASE permite ştergerea unor variabile nominalizate în <listă-variabile>. Dacă este prezentă opţiunea ALL, sunt şterse automat toate variabilele.

Clauza <masca> permite selectarea variabilelor care vor fi şterse (clauza LIKE) sau nu (clauza EXCEPT).

Exemplu:

RELEASE alfa, beta && se elimină din memorie variabilele alfa şi beta

RELEASE ALL LIKE a*

Note se înlătură din memorie toate variabilele care încep cu litera a

RELEASE ALL EXCEPT b?

NOTE se vor elimina toate variabilele cu excepţia celor al căror nume este format din două caractere, dintre care primul este b

Operația de ştergere a tuturor variabilelor poate fi realizată prin alte două comenzi:

CLEAR MEMORY **CLEAR ALL**

Folosirea distinctă a celor două comenzi RELEASE, CLEAR ALL este legată de statutul de variabilă publică sau privată a unei variabile.
Comanda RELEASE ALL şterge toate variabilele locale, dar nu acționează asupra celor publice.
Comanda CLEAR permite ştergerea variabilelor publice, care în mod firesc se şterg numai la încheierea sesiunii de lucru.

Salvarea şi restaurarea variabilelor

O altă problemă importantă este salvarea variabilelor create într-o sesiune de lucru ca variabile publice sau private, în vederea refolosirii lor ulterioare.
Trecerea pe disc a acestor variabile utile mai multor sesiuni într-un fişier special cu extensia .MEM se face prin comanda SAVE:

SAVE TO <fis.mem> [ALL LIKE /EXCEPT <masca>]

Sunt trecute pe disc în fişierul <fis.mem> fie toate variabilele (este opțiunea implicită) fie numai o parte a acestora (clauza ALL LIKE va indica cele ce se vor păstra, clauza ALL EXCEPT pe cele ce se vor ignora la salvare).
Exemplu:

```
CLEAR  ALL
CLEAR
a = 1
b = 2
suma = a + b
? a , " + " , b , " ? " , suma
```

1 + 2 = 3

```
SAVE TO fvar
```

NOTE   se salvează variabilele a , b şi suma în fişierul fvar.mem

a = 5
b = 3
suma = a + b
? a , “ + “ , b , “ ? “ , suma
5 + 3 = 8
RESTORE FROM fvar
? a , “ + “ , b , “ ? “ , suma
1 + 2 = 3

Restaurarea înseamnă trecerea variabilelor din fişier în memoria de lucru şi se face prin suprascriere prin comanda RESTORE:

RESTORE FROM <fis. mem> [ADDITIVE]

Clauza ADDITIVE este necesară atunci când vrem să se adauge variabilele salvate la cele existente.

O tehnică specială de lucru cu variabile o reprezintă macrosubstituția, prin care conținutul unei variabile de tip şir de caracter este tratat ca numele altei variabile. Macrosubstituția funcționează ca şi cum în locul variabilei respective ar fi pus şirul de caractere conținut de aceasta, fără apostrofurile delimitatoare. Sintaxa este:

& < var > [. < expC >]

în care <var> desemnează variabila de tip şir de caractere care va fi substituită de conținutul său.

Exemplu:

a = “ alfa “
alfa = “ Salut   ! “
? & a
Salut   !
? alfa && echivalentă cu comanda anterioară

Salut !

O altă metodă de referire indirectă la o variabilă dată este reprezentată de expresiile nume. Acestea determină tratarea valorii unei espresii ca un nume. Pentru ca o expresie să fie tratată ca o expresie de tip nume, aceasta se încadrează între paranteze rotunde. Construcția este tratată ca o expresie de tip nume numai acolo unde nu există posibilitatea confundării parantezelor rotunde cu cele care grupează operațiile din expresii.

Exemplu:

a = “ nume “
b = “ propriu “
? (a + b)   && nu va fi tratată ca o expresie numerică
nume propriu
……….
REPLACE (a) WITH “ Popescu”
NOTE în acest caz a este o expresie nume

Ori de câte ori este posibilă folosirea expresiilor nume, se recomandă această metodă deoarece macrosubstituția este mai lentă.

Afişarea listei variabilelor existente se realizează prin următoarea comandă:

DISPLAY/LIST MEMORY [TO PRINTER/TO FILE <fis. txt>]

Comanda permite trecerea în revistă a variabilelor utilizator cu statutul lor public sau privat, tipul şi valoarea în momentul respectiv. Statutul public al unei variabile îi permite să fie recunoscută în mai multe programe; altfel ea are caracter privat, fiind locală programului care a definit variabila.

Clauza TO PRINTER permite afişarea listei la imprimantă, iar clauza TO FILE trecerea listei de variabile într-un fişier text.

Masive

O variabilă poate memora la un moment dat o singură valoare, de un anumit tip, tipul variabilei respective. Pentru memorarea simultană a mai multor valori se pot folosi mai multa

variabile, cărora li se atribuie pentru identificare nume distincte. Dar ce facem atunci când numărul valorilor care trebuiesc memorate simultan este mare? O comanda care spre exemplu , ar incărca variabilele respective cu date dintr-un fişier, ar ocupa sute de linii din program, necesare pentru specificarea tuturor variabilelor care vor fi încărcate. Evident că acest lucru este ineficient , acestă metoda fiind nerecomandată.

O altă metoda pentru memorarea mai multor valori în memoria interna a calculatorului este oferită de masive , structuri de date care permit memorarea mai multor valori într-o zonă de memorie continuă căreia i se atribuie un nume, valorile respective putând fi tratate ca un tot unitar (în comenzi şiîn funcâii speciale) , cât şi independent, ca variabile simple.

Masivele sunt organizate sub forma unui tablou de valori, unidimensional sau bidimensional, deci sub formă de vector sau de matrice. Declararea unui masiv (vector sau matrice), presupune următoarele operații:

-stabilirea tipului masivului, adică dacă acesta este vector sau matrice;

-rezervarea zonei de memorie necesară depozitării valorilor care vor fi memorate în masiv, în funcție de numărul de elemente ale acestuia;

- atribuirea unui nume, prin caremasivul va fi identificat.

Declararea unui masiv se realizează prin una din comenzile DIMENSION sau DECLARE, care sunt identice ca funcționare şi sintaxa:

DIMENSION <masiv1> (<expN1> [,<expN2>])
[, <masiv2> (<expN3> [,<expN4>])] …

DECLARE <masiv1> (<expN1> [,<expN2>])
[, <masiv2> (<expN3> [,<expN4>])] …

cu aceste comenzi se pot declara unul sau mai multe tablouri, ale căror nume vor fi <masiv1>, <masiv2>,…….. Tipul tabloului, unidimensional (vector) sau bidimensional (matrice), va fi dat de numărul de expresii numerice care urmează numelui:

unidimensional, când este prezentă o singură expresie, şi bidimensional, când sunt prezente două valori numerice între parantezele rotunde.

Dimensiunea tabloului, adică numărul de elemente ale acestuia, este dată de valorile expresiilor dintre paranteze <expN1> şi <expN2> pentru primul tablou şi <expN3> şi <expN4> pentru cel de-al doilea şi aşa mai departe. Astfel tabloul unidimensional <masiv1> va avea <expN1> elemente numerotate de la 1 la <expN1> iar, dacă acesta este bidimensional, numărul de elemente va fi <expN1>*<expN2>.

În cazul unui tablou bidimensional, deci o matrice, se adoptă următoarea tehnologie, preluatade la lucrul cu matrice: prima expresie din paranteză va da numărul de linii ale matricei, numerotate de la 1 la <expN1> iar cea de a doua expresie numărul de coloane ale matricei, numerotate de la 1 la <expN2>.
Exemplu:

DIMENSION a(10)

NOTE defineşte vectorul a cu 10 elemente,numerotate de la 1 la 10.

DIMENSION alfa(2,4)

NOTE va declara matricea alfa cu 2 linii şi 4 coloane

DECLARE vector (3) , mat (5,10)

NOTE se defineşte vectorul vector cu 3 elemente şi matricea mat cu 5 linii si 10 coloane.

În comenzile DIMENSION şi DECLARE se pot înlocui parantezele rotunde cu cele pătrate, fără a afecta comanda respectivă.

Elementele masivului sunt identificate prin poziția acestora în cadrul tabloului, astfel:

- printr-un singur număr, indicând poziția elementuluiîn cadrul vectorului;
- prin doua numere,care vor desemna linia si coloana la care se afla elementul respectiv.

Exemplu: a (5) identifica al 5-lea element al vectorului a

Alfa (2,3) desemneaza elementul de pe linia 2 si coloana 3 a matricei alfa.

Elementele unei matrici pot fi, de asemenea, desemnate printr-o singură valoare numerică, care va indica poziția în matrice a elementului respectiv, numărarea acestora făcându-se în ordinea următoare: mai întâi se număra elementul unei linii, după care se trece la următoarea.

Exemplu: elementul alfa (2,3) poate fi identificat şi prin alfa (7):

alfa(1)	alfa(2)	alfa(3)	alfa(4)
alfa(5)	alfa(6)	alfa(7)	alfa(8)

alfa (2,3)

După declararea unui masiv toate elementele acestuia vor fi de tip logic, având inițial valoarea .F. Atât tipul cât şi valoarea unui element al masivului, pot fi schimbate printr-o instrucțiune de atribuire (operatorul de atribuire "=" sau comanda STORE).

Elementele unui masiv nu trebuie neapărat să aibă aceleşi tip (spre deosebire de alte limbaje, Pascal, C).

O modalitate specială de inițializare a elementelor unui masiv este dată de folosirea unei instrucțiuni de atribuire în care, în locul variabilei de atribuit, este introdus numele masivului. Astfet, toate elementele masivului vor capăta valoarea expresiei din instrucțiune.

Exemplu: DIMENSION a (10)
STORE 0 TO a
NOTE toate elementele masivului vor fi de tip numeric, avand ;
initial valoarea 0.
? ' a (3) =', a (3)
a (3) = 0
a = ' '
NOTE tipul tuturor elementelor masivului a va fi sir de caractere;
şi toate elementele acestuia vor avea valoarea ' '

Marimea si dimensiunile unui tablou creat anterior se pot schimba printr-o noua comanda DIMENSION sau DECLARE, prin redefinirea acestuia.

Se pot realiza astfel:

- marirea sau micşorarea dimensiunii unui tablou unidimensional ;
- transformarea unui tablou unidimensional într-unul bidimensional şi incers;
- redimensionarea unui tablou bidimensional.

Pentru masive unidimensionale:

- la mărirea numărului de elemente ale acestuia, vechile elemente vor rămâne neschimbate, iar noile elemente vor fi de tip logic, având inițial valoarea .F.;
- la micşorarea dimensiunii masivului, elementele care sunt în plus vor fi eliminate din memorie, iar celelalte vor rămâne neschimbate.

Transformarea unui masiv bidimensional într-unul unidimensional se face copiind elementele vechiului masiv, linie cu linie, în cadrul fiecărei linii copierea făcându-se de la primul până la ultimul element al acesteia. În această situație apar două cazuri:

- masivul bidimensional avea mai multe elemente decât noul masiv unidimensional: în acest caz restul elementelor care nu au încăput se pierd;
- noul masiv creat, unidimensional, are mai multe elemente decât noul masiv bidimensional, caz în care restul elementelor masivului unidimensional pentru care nu au mai fost elemente de copiat din primul masiv, vor fi inițializate cu valoarea logică .F.

Trecerea inversă se face în mod analog: completarea masivului bidimensional se face pe linii , începând cu prima linie a masivului şi terminând cu ultima. Şi în acest caz elementele care nu încap în noul masiv bidimensional se pierd, iar elementele masivului bidimensional care nu au corespondent în masivul unidimensional vor fi de tip logic, cu valoarea inițială .F.

Redimensionarea unui masiv bidimensional se realizează prin alocarea memoriei necesare noului masiv bidimensional, după

care urmează copierea elementelor vechiului masiv în cel nou, în ordinea numerotării acestora.

Elementele vechiului masiv bidimensional care nu încap în cel nou se vor pierde, iar dacă există elemente ale noului masiv care nu au corespondent în vechiul masiv, acestea vor fi inițializate la valoarea logică .F.

Exemplu:

```
        DIMENSION a ( 2,3 )
        FOR  I = 1 TO  6
                a ( I ) = 0
        ENDFOR
        FOR  I = 1 TO  2
                ?
                FOR  J = 1  TO  3
                        ?? a ( I,J )
                ENDFOR
        ENDFOR
        ?
    DIMENSION a ( 3,4 )
        FOR  I = 1 TO  3
                ?
                FOR  J = 1  TO  4
                        ?? a ( I,J )
                ENDFOR
        ENDFOR
```

Vom obține pe ecran următoarele rezultate:

```
1     2     3
4     5     6

1     2     3     4
5     6     .F.   .F.
.F.   .F.   .F.   .F.
```

Vom prezenta în continuare funcțiile referitoare la prelucrarea masivelor.

Numărul elementelor unui masiv, numărul liniilor sau numărul colanelor acestuia, se obține prin funcția ALEN ( ), aceasta având sintaxa:

ALEN (< masiv> [, < expN >])

< masiv > desemnează masivul asupra căruia ne informăm, iar <expN> este o expresie numerică ce determină informația returnată, astfel;

- când valoarea acesteia este 0, funcția returnează numărul de elemente ale masivului;
- la valoarea 1 a expresiei funcția va returna numărul de linii ale masivului (numărul de elemente pentru masive unidimensionale);
- în cazul valorii 2 a lui < expN>, funcția va returna numărul de coloane ale masivului (0 pentru masivele bidimensionale).

Valoarea returnată este de tip numeric.
Exemplu:

DIMENSION alfa (3,5)
? ‘ tabloul alfa are ‘ ,ALEN (alfa), ‘elemente’
Tabloul alfa are 15 elemente
? ‘Acest tablou are ‘ ,ALEN (alfa, 1), ‘ linii si ‘ , ALEN (alfa , 2) , ‘ coloane’
Acest tablou are 3 linii si 5 coloane

Absența expresiei numerice <expN> este echivalentă cu valoarea 0 a acesteia.
Exemplu:

DIMENSION a (10)
? ALEN (a) = ALEN (a, 0)
.T.

Înserarea unui element, a unei linii sau a unei coloane într-un masiv se realizează cu funcția AINS ( ), având sintaxa:

AINS (< masiv> [, 2])

Această funcție lucrează diferit, în funcție de numărul dimensiunilor masivului:

- pentru masive unidimensionale funția va însera un nou element în poziția <expN> a masivului <masiv>;
- pentru masive bidimensionale funcția va însera o linie, a <expN>-a linie a masivului <masiv>, când lipseşte parametrul 2 din apelul funcției, sau va însera o coloană, a <expN>-a coloană, în cazul când parametrul 2 este prezent.

Înserarea unui element, a unei linii sau a unei coloane într-un masiv nu va determina modificarea dimensiunii acestuia ci pierderea elementelor care nu mai încap în masiv după ce înserarea are loc.

Exemplu:

```
DIMENSION a ( 5 )
FOR  I = 1 TO  5
        a ( I ) = I
ENDFOR
?
FOR  I = 1 TO  5
        ? ? a ( I )
ENDFOR
AINS ( a , 3 )
FOR  I = 1 TO  5
        ? ? a ( I )
ENDFOR
```

În exemplul anterior, înserarea elementului nou pe poziția a 3-a a masivului unidimensional a va avea loc astfel:

a (1)	a (2)	a (3)	a (4)	a (5)
1	2	3	4	5
1	2	.F.	3	4	5

elementul nou apare aici

acest element se pierde

Funcţia opusă lui AINS ( ) este funcţia ADEL ( ), care şterge un elemen, o linie sau o coloană a unui masiv. ADEL ( ) are sintaxa:

ADEL (< masiv >, <expN> [, 2])

Funcţia ştergând elementul <expN> al masivului, în cazul când acesta este unidimensional, sau linia sau coloana <expN> a masivului, când acesta este bidimensional. Parametrul 2, ca şi la funcţia anterioară, face distincţia între ştergerea unei linii (absenţa acestui parametru) şi ştergerea unei coloane (prezenţa parametrului).

După ştergere, celelalte elemente ale masivului sunt translatate, în vederea umplerii golului format, iar pe ultima poziţie, ultimul element, ultima linie sau ultima coloană, eliberată prin translatare, se va introduce valoarea logică .F.

Exemplu:

```
 DIMENSION a ( 5 )
FOR  I = 1 TO  5
        a ( I ) = I
ENDFOR
?
FOR  I = 1 TO  5
        ? ? a ( I )
ENDFOR
ADEL ( 3 )
FOR  I = 1 TO  5
        ? ? a ( I )
ENDFOR
```

dispare acest element

a (1)   a (2)   a (3)   a (4)
a (5)
1   2   3   4
5
1   2   4   5
.F.

Referirea la elementele unui masiv bidimensional se poate face în două moduri: prin doi indici reprezentând linia şi coloana

elementului respectiv în cadrul masivului, sau printr-un singur indice, acesta reprezentând poziția elementului în masiv, numerotarea masivului făcându-se astfel: mai întâi se numără elementele primei linii, după care se trece şa cea de-a doua linie şi aşa mai departe.

Pentru a afla poziția unui element al unui masiv, când se cunoaşte linia şi coloana pe care se află acesta, vom folosi funcția AELEMENT ( ), care are următoarea sintaxă:

AELEMENT (<masiv> , <expN1> [, <expN2>])

<masiv> desemnează tabloul, masivul la care se referă funcția, iar <expN1> şi <expN2> reprezintă linia respectiv coloana elementului referit. Dacă <exp2> lipseşte, masivul este unidimensional, funcția returnând valoarea <expN1>. Rezultatul funcției este de tip numeric.

Unele funcții referitoare la masive manipulează elementele tablourilor bidimensionale printr-un singur indice, pentru aflarea acestuia folosindu-se funcția prezentată anterior. Funcția care realizează transformarea inversă , deci de la un singur indice la doi indici, este funcția ASUBSCRIPT ( ) :

ASUBSCRIPT (<masiv>, <expN1>, <expN2>)

<masiv> reprezintă tabloul la care se referă funcția, iar <expN1> reprezintă poziția elementului în tablou, <expN2> este o expresie numerică ce determină tipul informației returnate de funcție:

- linia elementului, când <expN2> are valoarea 1;
- coloana elementului, în cazul valorii 2 a lui <expN2>.

Exemplu: având un masiv bidimensional definit cu :

DIMENSION a (4, 6)

Următoarele referiri sunt echivalente:

a (2,3),
a (9),
a (AELEMENT (a, 2, 3)),
a (ASUBSCRIPT (a, 9, 1), ASUBSCRIPT (a, 9, 2))

Copierea elementelor unui masiv în elementele altui masiv se face cu ajutorul funcției ACOPY ( ):

ACOPY (<masiv1>, <masiv2>, [, <expN1> [, <expN2> [, <expN3>]]])

Funcția va determina copierea a <expN2> elemente ale masivului <masiv1> începând de la al <expN1>-lea inclusiv, în elementele masivului<masiv2>, începând de la poziția <expN3>. Observăm că referirea la elementele masivului se face printr-un singur indice. Dacă <expN1> lipseşte, se presupune implicit valoarea 1 a acestei expresii.

Pentru a copia toate elementele masivului <masiv1>, începând de la al <expN1>-lea şi până la sfârşit, în masivul <masiv2>, vom folosi valoarea -1 pentru expresia <expN2>.

Când masivul <masiv2> nu există, se creează unul nou, cu aceleaşi dimensiuni în care se copiază conținutul masivului <masiv1>.

Căutarea unei expresii într-un masiv se face prin funcția ASCAN ( ) cu sintaxa:

ASCAN (<masiv>, <expr>, [expN2]])

în care <masiv> desemnează tabloul în care se caută expresia <expr>. Funcția returnează poziția elementului în care s-a găsit această expresie, în caz de reuşită, sau valoarea 0, când căutarea nu s-a încheiat cu succes.

Funcția ASORT ( ) sortează elementele masivului <masiv> în ordine crescătoare sau descrescătoare. Toate elementele sortate trebuie să fie de acelaşi tip, pentru a se putea compara între ele.

ASORT (<masiv> [, ,expN1 [, <expN2> [, <expN3>]]])

Pentru masivele unidimensionale se vor sorta elementele acestuia, iar pentru cele bidimensionale se vor sorta liniile masivului respectiv, în sensul că, în funcție de rezultatul comparării a două elemente ale unor linii distincte , se vor schimba sau nu între ele

toate elementele acestor linii, fiecare element rămânând pe coloana pe care a fost şi înainte.

Dacă masivul este unidimensional <expN1> va determina elementul de unde se începe sortarea (se vor srta elementele de la al <expN1>-lea încolo). În cazul când avem un masiv bidimensional, prima linie care va intra la sortare va fi cea pe care se află elementul al <expN1>-lea al masivului. De asemenea, <expN1> determină şi coloana de pe care se iau elementele de comparat, pentru a stabili ordinea liniilor.

Exemplu:

Dacă avem masivul a (3,4), iar <expN1> este 7, avem situația:

a (1,1)	a (1,2)	a (1,3)	a (1,4)
a (2,1)	a (2,2)	a (2,3)	a (2,4)
a (3,1)	a (3,2)	a (3,3)	a (3,4)

Elementul al 7-lea este a (2,3), deci se vor sorta liniile 2 şi 3 ale masivului, compararea acestora făcându-se prin a (2,3) şi a (3,3) (deci elementele de pe coloana a 3-a).

-<expN2> specifică numărul elementelor sortate, în cazul unui masiv unidimensional, sau numărul liniilor de sortat, pentru un masiv bidimensional. Dacă valoarea lui <expN2> este -1 sau dacă această espresie lipseşte, se vor sorta elementele, liniile, până la sfârşitul masivului (ultimul element, respectiv ultime linie).

-<expN3> determină ordinea sortării:

-crescătoare, dacă <expN3> lipseşte sau dacă acestă expresie are valoarea 0;

-descrescătoare pentru o valoare diferită de 0 a expresiei <expN3>.

Rezultatul funcției este de tip numeric.

Tipuri de date şi funcții standard

Datele cu care lucrează FoxPro sunt de tip numeric, caracter, data calendaristică, logic. Asupra acestor tipuri de date s-au definit

operații specifice şi au fost realizate funcții standard dintre care cele mai des folosite vor fi explicate în continuare.

Funcții uzuale asupra tuturor tipurilor de date:

MAX (<e1>,<e2>)	calculează maximul dintre două valori <e1> şi <e2>
MIN (<e1>,<e2>)	calculează minimul dintre două valori <e1> şi <e2>
TYPE(<eC>)	întoarce litera corespunzătoare tipului de dată.
IIF(<eL>,<e1>,<e2>)	întoarce <e1> dacă <eL> este adevărat şi <e2>în caz contrar

Tipul numeric

O mare parte a datelor prelucrate de calculator este reprezentată de numere, pentru a căror descriere se foloseşte tipul numeric. Cu toate că limbajul FoxPro este un limbaj orientat pe lucrul cu baze de date şi nu unul orientat pe calcule matematice, ştiințifice, tipul numeric este imlementat astfel ăcât să permită realizarea majorității operațiilor matematice ăntâlnite în practică.

De asemenea, sunt prevăzute o serie de funcții matematice prin care se pot calcula funcțiile matematice elementare.

Operanzii numerici care intervin în expresii pot fi:

- câmpuri numerice ale unei baze de date;
- funcții care returnează valori numerice;
- variabile de tip numeric;
- constante numerice.

Operatori care se aplică unor operanzi numerici, având ca rezultate tot valori numerice sunt : **, ^ (ridicarea la putere), * (înmulțire), / (împărțire), % (modulo, restul împărțirii), + (adunare), - scădere. Între două expresii numerice se pot aplica, de asemenea, operatori relaționali, obținându-se expresii logice.

Funcțiile standard uzuale:

ABS (<eN>)	calculează valoarea absolută din <eN>
SQRT (<eN>)	calculează radical din <eN> (strict pozitiv)
ROUND (<eN1>,<eN2>)	<eN1> este rotunjită la zecimala dată de <eN2>
MOD (<eN1>,<eN2>)	calculează restul împărțirii întregi a lui <eN1> la <eN2>
INT (<eN>)	întoarce un întreg rezultat prin trunchierea zecimalelor
CEILING (<eN>)	întoarce cel mai mic întreg mai mare sau egal cu argumentul <eN>
FLOOR (<eN>)	întoarce cel mai mare întreg mai mic sau egal cu argumentul <eN>
SIGN (<eN>)	întoarce valoarea –1 pentru argument negativ, 1 pentru argument pozitiv şi 0 pentru argument nul.

RAND ()	returnează un număr aleator în intervalul (0, 1)
STR (<eN1>[,<eN2> [,<eN3>]])	conversia între tipul numeric şi tipul şir: <eN1> este numărul, <eN2> este lungimea, <eN3> numărul de poziții pe care se va face reprezentarea părții zecimale.

Exemplu:

? MOD (38, 6)
2
? MOD (44,44 , 11,11)
0

Observație: Lista funcțiilor standard cuprinde şi funcții trigonometrice, logaritmi, radical, funcția exponențială, pe care le vom prezenta în anexă.

Fixarea numărului de zecimale pentru afişarea numerelor se poate face cu comanda SET DECIMALS:

SET DECIMAL TO <nr>

Exemplu:

? 2 / 3
0, 67
SET DECIMAL TO 4
? 2 / 3
0 , 6667

Exemple cu funcții:

? ABS (a)
400
? SIGN (- 32)
- 1
a = - 2 / 3
? a = SIGN (a) * ABS (a)
. T .
? INT (14 . 46)
14
? INT (- 2 . 25)
- 2
a = 14 . 46
? a – INT (a)
0 . 46
a = - 2 . 25
? a – INT (a)
- 0 . 25
? CEILING (8 . 32)
9
? CEILING (-4 . 23)
- 4
? FLOOR (8 . 32)
8

? FLOOR (- 4 . 23)
- 5
? EXP (2)
7 . 39
? LOG (2)
0 . 69
? LOG 10 (2)
1 . 00
? EXP (LOG (3))
3 . 00
? SQRT (2)
1 . 41

Funcțiile financiare:

Între funcțiile matematice, o categorie aparte o formează funcțiile FV(.,.,.), PV(.,.,.) şi PAYMENT(.,.,.) numite funcții financiare. În termeni economici funcția FV(.,.,.) calculează valoarea de viitor a unei investiții ("Future Value") iar funcția PV(.,.,.) calculează valoarea prezentată a unei investiții ("Prezent Value"). Detaliind:

1) FV(<expN1>, <expN2>, <expN3>) calculează valoarea viitoare a unei depuneri regulate cu o creştere constantă în cadrul unei investiții, cu o dobândă fixă pe o perioadă dată. Parametrii: <expN1> este depunerea, <expN2> este dobânda, <expN3> este perioada sau numărul de depuneri.

2) PV (<expN1>,<expN2>,<expN3>) calculează valoarea la zi a unei investiții constituite printr-un vărsământ regulat cu o sumă constantă, de-a lungul unui număr de perioade date şi când se practică o dobândă fixată. Parametrii funcției: <expN1> este suma plătită, <expN2> este dobânda, <expN3> este durata.

3) PAYMENT (<expN1>,<expN2>,<expN3>) calculează mărimea rambursărilor constante efectuate la intervale regulate care permit amortizarea unei sume, cu dobândă constantă, pe un număr dat de perioade. Parametrii: <expN1> este mărimea sumei cheltuite, <expN2> este taxa sau dobânda, <expN3> este numărul de rambursări.

Tipul şir de caractere

Un şir de caractere reprezintă o mulțime ordonată de caractere care se tratează ca un tot unitar. Într-un şir de caractere ordinea acestora fiind esențială, fiecărui caracter I se poate asocia un număr reprezentând poziția aceastuia în cadrul şirului.

Numărul caracterelor dintr-un şir reprezintă lungimea şirului. Un subşir al şirului dat repreuintă o porţiune din şir, începând de la o poziţie specificată şi de lungimea dată

Constantele de tip şir de caractere se specifică prin mulţimea caracterelor care le compun, încadrate între apostrofuri simple sau duble (la ambele capete trebuie să fie acelaşi tip de apostrof).

Pentru a include unul dintre cele două delimitatoare într-un şir de caractere, mulţimea caracterelor ce alcătuiesc şirul va fi încadrată între delimitatorul de celălalt tip decât cel din şir. Dacă lungimea şirului este 0 obţinem şirul vid sau nul, care se specifică prin două apostrofuri consecutive fără spaţii sau alte caractere între ele.

Datele de tip şir de caractere pot avea lungimea maxim 255 caractere ASCII şi se reprezintă intern câte un caracter pe octet în binar.

Operaţii asupra datelor de tip şir:

- concatenarea a două şiruri se realizează prin operatorii de concatenare (+, -); (+) realizează concatenarea a două şiruri;

Exemplu: “ strada _ “ + “ George _ Coşbuc “
După evaluare, va avea valoarea:
“ strada _ George _ Coşbuc “
Operatorul (-) realizează concatenarea termenilor cu mutarea spaţiilor de la sfârşitul primului şir la sfârşitul şirului rezultat.
Exemplu: “ Salut “ - “ prieteni !“
După evaluare, vom obţine şirul de caractere
“ Salut prieteni ! “
se observă că blancurile de la începutul şirului al doilea îşi păstrează poziţia în şir.

- testarea apartenenţei unui şir la un alt şir este realizată prin operatorul ($); poate fi folosit în expresii logice.

De exemplu, expresia:
“ calcul “ $ “ calculator “
este adevărată, pe când expresia:
“ calcule “ $ “ calculator “
va fi evaluată la valoarea logică fals.

Compararea şirurilor de caractere de lungimi diferite este controlată de comanda :

SET EXACT ON / OFF

În starea ON şirurile se compară de pe toată lungimea lor (cu excepţia spaţiilor de la sfârşitul şirului). În starea OFF se ia lungimea cea mai scurtă şi, dacă pe aceeaşi lungime şirurile sunt egale, rezultatul este .T..

Funcţii uzuale asupra şirurilor

SUBSTR (<eC>, <eN1>,<eN2>)	extrage un subşir din şirul <eC> începând cu caracterul de pe poziția <eN1> pe lungime <eN2>
LEFT (<eC>, <eN>)	extrage primele <eN> caractere din şirul <eC>
RIGHT (<eC>, <eN>)	extrage ultimele <eN> caractere din şirul <eC>
LEN (<eC>)	întoarce lungimea şirului <eC>
REPLICATE (<eC>, <eN>)	întoarce un şir având <eC> multiplicat de <eN> ori
SPACE (<eN>)	Întoarce un şir de <eN> spații
LTRIM (<eC>)	elimină spațiile de la stânga şirului <eC> ex.: LTRIM(’MIA’)=”MIA”
RTRIM (<eC>)/ TRIM (<eC>)	elimină spațiile de la dreapta şirului <eC> ex.: RTRIM(“MAI’”)=”MAI”
AT (<eC1>, <eC2>)	întoarce pozițiile şirului <eC1> în <eC2>
ISALPHA (<eC>)	testează dacă şirul începe cu o literă
ISLOWER (<eC>)	testează dacă şirul începe cu minusculă
ISUPPER (<eC>)	testează dacă şirul începe cu majusculă
LOWER (<eC>)	transformă şirul în minuscule
UPPER (expC)	transformă şirul în majuscule

STUFF (<eC1>, <eN1>, <eN2>, <eC2>)	înlocuieşte în <eC1> începând cu poziția <eN1> un subşir de lungime <eN2> prin şirul <eC2>
CTOD (<eC>)	realizează conversia unui şir la data calendaristică
VAL (<eC>)	realizează conversia unui şir la număr

Exemple:

? CHR (49)

1

? CHR (65) == “ A “

. T .

? ASC (“ A “)

65

? ASC (“ a “) = ASC (alfa)

. T .

? “ A “ == CHR (ASC (“ A “))

. T .

? 65 == ASC (CHR) 65))

. T .

? SUBSTR (“ ABCDEF “ , 2, 3)

BCD

? SUBSTR (“ Ziua Bună “ , 6)

Bună

? LEFT (“ La mulți ani ! “ , 2)

La

? RIGHT (“ Noapte bună ! “ , 6)

bună !

? REPLICATE (“ a “ , 5)

a a a a a

? REPLICATE (“ “ , 6) == SPACE (6)

. T .

? ALLTRIM (“   GAMA “) == “ GAMA “

. T .

? “ Mă numesc   “ + RTRIM (“ Ionescu   “) + “   Daniel “

Mă numesc Ionescu Daniel

? “ şi am “ + LTRIM (“   24 “) + “ ani. “
şi am 24 ani
? AT (“ nr. “ , “ Strada George Coşbuc, nr. 63 – 64 “)
22
? LEN (“ Salutări ! “)
10
? LEN (“ Strada George Coşbuc “ + “ nr. 150 “)
27
a = “ ALFA “
b = “ alfa “
? UPPER (a) == UPPER (b)
. T .
? LOWER (a) == LOWER (b)
. T .
STORE “ pala “ TO şir
şir = STUFF (şir , 3 , 0 , “ rale “)
? şir
paralela
şir = STUFF (şir , 3 , 3 , “ sar “)
? şir
pasarela
şir = STUFF (şir , 7 , 2 , “ “)
? şir
pasare

Tipul dată calendaristică

Datele calendaristice pot fi reprezentate în mai multe formate având ca delimitator acolada. Forma de prezentare a unei date calendaristice depinde de comanda SET DATE:

SET DATE [TO] <format>

unde <format> poate fi: AMERICAN / GERMAN / ANSI / ITALIAN / DMY /BRITISH /JAPAN /FRENCH /USA /MDY /YMD
Formatul AMERICAN prezintă data calendaristică sub forma: ll/zz/aa.; formatul GERMAN sub forma zz.ll.aa., etc.

Includerea secolului în formatul de dată este determinată de starea comutatorului SET CENTURY ON/OFF. Implicit este OFF.

Indicarea semnului folosit ca separator al informațiilor de tip dată calendaristică este dat de comanda SET MARK. Implicit separatorul este dat de formatul de reprezentare. De exemplu formatul AMERICAN foloseşte ca separator ”/”.

SET DATE TO <car>

unde <car> reprezintă un singur caracter ce va fi folosit ca separator al informațiilor din data calendaristică.

Operații care se pot face cu datele calendaristice sunt:

- compararea a două date se realizează prin operatorii relaționali:
- diferența dintre două date calendaristice dă un număr de zile:
- adunarea unui număr de zile la o dată calendaristică dă o altă dată:
- scăderea unui număr dintr-o dată calendaristică dă tot o dată;

Funcțiile referitoare la date calendaristice sunt :

DATE()	întoarce data curentă de la sistem
DAY (<eD>)	extrage nr. zilei din dată
MONTH(<eD>)	extrage nr. lunii din dată

CMONTH(<eD>)	întoarce numele lunii
YEAR(<eD>)	extrage anul din data calendaristică
TIME()	extrage ora sistem sub forma şirului 'HH:MM:SS'
DTOS(<eD>)	întoarce data sub forma 'secol – an lună zi'
DMY(<eD>)	întoarce data sub forma 'zi nume-lună an'
MDY(<eD>)	întoarce data sub forma 'nume – lună zi an'
DTOC(<eD>)	conversie data la şir

Exemple:

? DATE ( )
11 / 12 / 2000
? CDOW (DATE ( ))
Saturday
? DOW ({10 / 02 / 1864})
1
? DAY ({03 / 14 / 1990})
14
? MONTH (DATE ( ))
3
? CMONTH ({03 / 25 / 1990})
March

Operații elementare asupra bazelor de date

Crearea şi manipularea structurii conceptuale

Definirea structurii conceptuale a bazei de date este o operație foarte importantă, de care poate depinde întregul proiect al aplicației.

Proiectarea structurii logice porneşte de la “IEŞIRI”, adică de la cererile de informații, de la rezultatele pe care aplicația informatică trebuie să le furnizeze decidenților. După inventarierea tuturor cererilor de informații se determină “INTRĂRILE” adică datele care pot fi reținute într-o bază de date.

În general, se evită modificarea structurii conceptuale; de aceea administratorul bazei de date, cel care face proiectarea structurii, trebuie să analizeze nu numai cererile prezente de informații, pe care aplicația informatică trebuie să le ofere, cât şi cererile posibile în viitor sau solicitate accidental.

Structura conceptuală este ansamblul câmpurilor cu denumirea, lungimea şi tipul lor, precum şi ordinea de definire a acestor câmpuri.

Crearea structurii se realizează cu comanda CREATE

CREATE <fis.dbf>

Comanda CREATE permite deschiderea unui ecran de proiectare a structurii bazei de date şi, prin dialog cu utilizatorul, defineşte structura unei baze de date: numele fiecărui câmp, tipul, lungimea sa, numărul de zecimale şi dacă respectivul câmp va fi cheie de indexare într-un fişier multiindex asociat bazei de date.

Utilizatorul poate introduce date imediat după salvarea structurii răspunzând afirmativ la întrebarea sistemului: input data records now?(y/n)

La un răspuns “Y” se deschide ecranul de introducere a datelor (un ecran standard) în care apare pe linii câmpurile din structură cu numele lor şi, alăturat, o zonă invers video cu lungimea egală cu a câmpului a cărei valoare o va cuprinde.

Observație:

Dacă se iese accidental din ecranul de introducere şi mai sunt articole de adăugat, se poate folosi comanda APPEND.

Exemplu: creerea bazei de date mijloacef.dbf, memorand starea mijloacelor fixe ale unei unitati economice, cu structura:

COD character 10 codul mijlocului fix

DENUMIRE character 30 denumirea
VALOARE numeric 10 valoarea
AMORTIZARE numeric 10 valoarea amortizata
LOC_FOLOS memo 10 locul de folosinta
STARE logical 1 starea (în folosinta .T. , nefolosit .F.)
DATA_INST date 8 data instalarii, puneriiîn functionare
se realizeaza cu comanda:

```
CREATE  TABLE mijloacef;
        (  cod C (10), denumirea  C (30),  valoare  N(10), amortizare N(10),;
        loc_folos M,  stare  L, data_inst D )
```

Deschiderea şi închiderea bazei de date

Orice operație, cu excepția creerii bazei de date, presupune deschiderea acesteia iar după terminarea activității asupra datelor respective bazele de date trebuiesc închise.
Comanda de **deschidere** a bazei de date este USE:

USE<fis.dbf>

Comanda USE deschide baza de date de nume specificat închizând, eventual, o altă bază de date dacă este deschisă. Extensia este implicită.

Comanda de **închidere** a bazei de date deschisă anterior este:
USE

Exemplu.
use mijloacef &&& s-a deschis baza de date **mijloacef**
use produse && s-a închis baza **mijloacef** şi s-a deschis baza **produse**
use && s-a închis baza de date produse

Zone de lucru

Sunt multe aplicații care necesită accesul simultan la mai multe baze de date. Pentru aceste situații se folosesc zone distincte de memorie numite **zone de lucru**. Într-o zonă de lucru se poate deschide o singură bază de date. În diferite variante ale pachetului de programe dBASE numărul de zone variază:

-in dBASE 4 sunt 10 zone de lucru identificate prin numere de la 1 la 10 sau literele A-J, -în FoxPro sunt 25 zone de lucru identificate cu numerele 1-25 sau cu literele alfaberului A-J (pentru primele 10) şi W11-W25 (pentru zonele 11-25).

Indicarea zonei de lucru unde se vor desfăşura următoarele operații se face prin:

SELECT <zona> / <nume-alias>

Comanda funcționează ca un comutator pe zona dată prin construcția <zona> (ce poate fi număr sau literă asociată zonei de lucru) sau prin construcția <nume-alias> care este o prescurtare a numelui de fişier deschis în zonă.

Deschiderea unei baze de date într-o zonă se poate face prin comanda USE cu o clauză nouă, clauzaîn <zona>. Tot prin comanda USE se poate asocia şi un alias (un pseudonim, o prescurtare) în vederea unei referiri mai clare atât a câmpurilor cât şi a zonei în care s-a deschis fişierul.

USE <fis.dbf>în <zona> [ALIAS <nume-alias>]

Calificarea câmpurilor: Atunci când sunt deschise mai multe fişiere, referirea la câmpurile lor se poate face printr-o construcție de forma:
<zona>/<nume-alias> <separator> <nume-câmp>,
unde <zona> este litera asociată zonei unde s-a deschis fişierul; <separator> este format din semnele -> (minus şi mai mare). Exemplu: A->nume 0

Comenzile de poziționare ca şi funcțiile asupra bazelor de date pot avea referire directă în zona cercetată. Astfel clauzaîn poate completa comenzile:

GOTO/GO/GO TOP/GO BOTTOM in <zona>

SKIP [+/-]<n>în <zona>

Aliasul fişierului poate să apară şi în funcțiile care lucrează asupra fişierelor:

1) EOF(<zona>) && testează sfârşitul de fişier din zona <zona>

2) BOF(<zona>) && testează poziția înaintea primului articol
&& din zona <zona>

3) RECNO(<zona> && dă numărul articolului curent din fişierul
&& deschis în zona <zona>

Funcții necesare:

SELECT () && dă primul număr de zonă liber de folosit.

Observație importantă:

Zonele de lucru sunt izolate; modificarea pointerului de înregistrare ca urmare a unei acțiuni într-o bază de date nu poate determina modificarea pointerului unei alteia, deschisă în altă zonă de lucru.

Fac excepție de la această regulă fişierele înlănțuite cu
SET RELATION.

Exemplu:

```
? SELECT ( )      && afiseaza zona de lucru curenta
1
USE mijloacef
NOTE s-a deschis baza de date mijloacefîn zona de lucru 1
USE     && s-a inchis baza de date
USE mijloacef în 2
```

NOTE s-a deschis baza de dateîn zona de lucru 2, chiar daca zona curenta;
de lucru a fost si va ramane 1
USEîn 2 && se inchide baza de date din 2

Observaţie: în FoxPro o aceeaşi bază de date se poate deschide în zone diferite folosind în acest scop clauza AGAIN în comanda de deschidere.
Exemplu:

```
SELECT  a
USE mijloacef  în 1
USE mijloacef  AGAIN
? USED ( )    && testeaza folosirea zonei de lucru 1
.T.
? USED (2 )    && testeaza folosirea zonei de lucru 2
.F.
?  USED ( 'mijloacef' )
NOTE testeaza  daca exista baza de date  cu aliasul mijloacef;
 intr-o zona de lucru
.T.
USE
USEîn 1
```

Modificarea structurii bazei de date

Comanda MODIFY STRUCTURE permite modificarea structurii unei baze de date.

MODIFY STRUCTURE

Comanda permite accesul utilizatorului la structura bazei de date active deschisă în acel moment în zona de lucru. Utilizatorul poate şterge, adăuga, sau insera câmpuri, poate modifica lungimea, sau tipul unor câmpuri. Datele existente se vor copia în noua

structură prin verificarea numelui de câmp din cele două structuri. Dacă coincid, datele se vor trece pe noua structură făcându-se conversia, acolo unde este posibil, la noul tip de câmp.

Atenție! Dacă am schimbat lungimea câmpului s-ar putea ca valorile existente în structura anterioară să nu "încapă" şi, dacă se va face trunchiere, la numere se vor pierde valori (apar steluțe!). Dacă împreună cu adăugarea unor câmpuri se va face şi schimbarea numelui unor câmpuri, se vor pierde date.

Afişarea structurii

LIST/DISPLAY STRUCTURE [TO PRINTER/TO FILE <fis.txt>]

Comenzile LIST şi DISPLAY sunt asemănătoare, cu deosebirea că DISPLAY face o pauză la umplerea unui ecran. Afişarea structurii poate fi direcționată la imprimantă sau într-un fişier text.

Exemplu:

```
SELECT a
USE mijloacef
LIST STRUCTURE
```

Structure for database: C:\FOXPRO\MIJLOACAF.DBF
Number of data records: 6
Date of last update: 02/22/95

Field	Field Name	Type	Width	Dec	Index
1	COD	Character	10		Y
2	DENUMIRE	Character	30		N
3	VALOARE	Numeric	10	0	N
4	AMORTIZARE	Numeric	10	0	N
5	LOC_FOLOS	Memo	10		N
6	STARE	Logical	1		N
7	DATA_INST	Date	8		N
8	Tip	Character	1		N
** Total **			81		

Duplicarea structurii conceptuale:

COPY STRUCTURE TO <fis.dbf> [FIELDS<lista-camp>]

Comanda permite crearea unei noi baze de date numită <fis.dbf> pornind de la baza de date activă, prin preluarea tuturor câmpurilor (dacă lipseşte clauza FIELDS) sau a anumitor câmpuri enumerate în clauza FIELDS.

Exemplu: din baza de date mijloacef se creaza o noua baza de date, mijloacaf_n,în care vom copia doar campurile COD, DENUMIRE si VALOARE:

```
select a
use mijloacef                 && se deschide baza de date mijloacef
copy structure to mijloacef_n fields cod, denumire, valoare
use mijloacef_n
list structure
Structure for database:            C:\FOXPRO\MIJLOACAF.DBF
Number of data records:            6
Date of last update:               02/22/95
Field  Field Name      Type       Width   Dec    Index
1      COD             Character  10             Y
2      DENUMIRE        Character  30             N
3      VALOARE         Numeric    10      0      N
** Total **                       51
```

Există încă o modalitate de a crea structura unei baze de date: CREATE TABLE care permite specificarea directă, în comandă, a structurii. Are formatul:

CREATE TABLE<fis.dbf> (<lista-definiții>)

unde <lista-definiții>:=<nume-câmp> <tip>[([<lungime>], <zecimale>])]
Construcția <tip> este una din literele asociate tipurilor de date.

Funcții standard relativ la structura bazei de date:

1) FIELD(<expn>) întoarce numele câmpului din baza de date activă care are numărul de ordine <expn>;

2) RECSIZE() întoarce dimensiunea în octeți a structurii bazei de date active;

3) TYPE(<câmp>) întoarce tipul unui câmp precizat ca şir de caractere;

4) FLDCOUNT() întoarce numărul de câmpuri din structura bazei de date.

Exemple:

use mijloacef && deschidere fişier mijloacef.dbf

?field(1) && se afişează numele primului câmp

COD

?field(6) && se afişează numele celui de-al 6-lea câmp

STARE

?field(15) && dacă numărul dat ca parametru depăşeşte numărul de

&& câmpuri din structură, funcția întoarce şirul vid

?field(15)=” && testăm dacă rezultatul întors de funcție este şirul vid

.T.

?field(15)=’’ && şirul vid este diferit de caracterul spațiu “ ”

.F.

use mijloacaf_N

?recsize()

51 && se observă afişarea dimensiunii articolului

?type(“denumire”)

C && câmpul nume are tipul caracter

use mijloacef

?fldcount() && fişierul mijloacef are în structură 8 câmpuri

.8.

use mijloacef_n

?fldcount() && fişierul PROBE are 3 câmpuri

3

Structura fizică a unei baze de date cuprinde ansamblul valorilor câmpurilor grupate în înregistrări sau articole. Articolele se depun în tabelă sau bază de date unul în continuarea celuilalt şi fiecare poartă o informaţie (un număr), ce se asociază automat la introducerea valorilor articolului în fişier. Numerele sunt în secvenţă strict crescătoare, determinând poziţia fizică a articolului în fişier. Numărul de articol permite regăsirea rapidă a înregistrării. În fiecare moment al prelucrării unui fişier se păstrează numărul articolului prelucrat (articolul curent) în pointerul de fişier sau indicatorul de înregistrare.

Fişierul are un marcator de început şi un marcator de sfârşit, între care se poate opera cu înregistrările utilizator.

Câteva funcţii necesare prelucrării articolelor:

a) RECNO () && întoarce numărul articolului curent

b) EOF () && întoarce .T. dacă în urma prelucrărilor s-a ajuns
&& la sfârşitul de fişier şi .F. în caz contrar.

c) BOF () && întoarce .T. dacă în urma prelucrărilor s-a ajuns
&& înaintea primului articol cu date şi .F. în caz contrar.

d) RECCOUNT () && întoarce numărul de articole din baza de date

Selectarea articolelor

Prelucrările asupra unei baze de date (afişări, copieri, ştergeri, modificări etc.) pot fi realizate pe toate articolele bazei sau pe o parte a acestora.

Operaţia de selectare a articolelor care vor fi prelucrate poate fi indicată chiar în comanda de prelucrare prin clauzele de selectare.

<domeniu>, **FOR <COND>** **WHILE <COND>**

Clauza <domeniu> poate fi înlocuită cu următoarele cuvinte cheie:

ALL: sunt selectate roate articolele fişierului,

NEXT <n> sunt selectate următoarele <n> articole față de articolul curent;
REST: sunt selectate toate articolele până la sfârşitul fişierului începând cu articolul pe care am fost poziționați anterior acestei comenzi;
RECORD <n>semnifică articolul cu numărul <n>.

Clauza **FOR <cond>** permite selectarea articolelor care verifică condiția indicată prin construcția <cond>, implicit sau prin testarea acesteia pe întreg domeniul precizat în comandă.

Clauza **WHILE <cond>** selectează articolele care verifică condiția dată; această verificare încetează atunci când se găseşte primul articol care dă ca rezultat al condiției valoarea .F. (fals).

Adaugarea de inregistrări la o baza de date

Completarea cu valori a articolelor bazei de date se poate face imediat cu proiectarea structurii conceptuale, prin răspunsul "y" la mesajul sistem afişat sau ulterior, prin comanda APPEND.

APPEND

Comanda APPEND este o comandă (mod ecran) care permite introducerea datelor de la tastatură. Standard, acest ecran prezintă pe o linie câte un câmp cu numele lui şi, în continuare, marcat invers-video, zona de introducere. Utilizatorul va introduce valorile articolului, câmp după câmp şi va trece automat la completarea articolului următor odată cu terminarea completării ultimului câmp.

Observații:

Completarea câmpurilor memo se face prin apăsarea simultană a tastelor <ctrl><home>. Se deschide o fereastră de editare unde se poate introduce textul asociat câmpului memo, iar la sfârşitul editării se apasă pe tastele <ctrl><end> (salvare) sau <esc> (abandon).

Datele introduse în zonele invers-video trebuie să aparţină tipului declarat la definirea structurii pentru câmpul respectiv. Se verifică de asemenea şi încadrarea valorii câmpului în lungimea declarată.

Semnalizarea cazurilor de eroare se face, de regulă, sonor. Există o comandă comutator SET BELL ON/OFF care determină emiterea semnalului sonor de avertizare (ON) sau inhibă această semnalizare (OFF).

Comanda comutator SET CONFIRM ON/OFF aşteaptă confirmarea introducerii valorilor într-un câmp prin tasta <enter> (ON) sau trecerea automată la completarea valorilor câmpului următor imediat ce dimensiunea unui câmp a fost depăşită (OFF). Implicit este pe OFF.

Verificarea apartenenței la tipul dată calendaristică se face în funcție de setarea existentă.

Comanda SET DATE dă posibilitatea setării în sesiunea curentă a formatului de dată. Peste tot unde se lucrează cu date calendaristice ele vor fi "văzute" în formatul respectiv. Deci dacă a fost introdusă anterior o comandă SET DATE BRITISH (zz/ll/aa) se va verifica condiția ca primele 2 cifre să fie încadrate în intervalul 1..31, următoarele cifre între 1..12. Alta va fi validarea aceleiaşi valori introduse în câmp dacă anterior era setarea în format american (ll/zz/aa).

O facilitate la introducerea interactivă a datelor este oferită de comanda SET CARRY ON/OFF. Atunci când introducem seturi de date grupate, unele informații se vor repeta la mai multe articole.

De exemplu la toate facturile unui partener, numele, adresa, codul fiscal al partenerului sunt aceleaşi şi totuşi aceste date trebuiesc trebute în fiecare articol.

SET CARRY ON copiază în articolul curent valorile articolului precedent.

Implicit, cumpărătorul este pe valoarea OFF; în această situație fiecare nou articol va avea câmpurile vide.

Un câmp este vid dacă are spații (caracter), zero (numeric), .F. (logic) sau {//} (dată calendaristică).

Vizualizarea conținutului unei baze de date

Afişarea informațiilor conținute într-o bază de date este esențială pentru utilizator. O modalitate de afişare este oferită de comenzile DISPLAY şi LIST

LIST/DISPLAY[<listaexpr>][<domeniu>][FOR<cond>] [WHILE<cond>][TO PRINTER/TO FILE <fis.txt>] [OFF]

În absența oricărei clauze, comanda LIST va afişa întreaga bază de date, iar DISPLAY doar un singur articol: cel pe care ne-am poziționat ca urmare a unor manevre anterioare.

Clauzele <domeniu>, FOR, WHILE permit selectarea articolelor ce vor fi afişate. Clauza <list-exp> enumără fie câmpurile, fie expresii care se vor afişa cu aceste câmpuri. În lipsa acestei clauze se vor afişa toate câmpurile. Clauzele TO PRINTER permite trimiterea conținutului fişierului la imprimantă iar TO FILE <fis.txt> permite trimiterea conținutului bazei de date într-un fişier text. Clauza OFF inhibă afişarea numărului de articol dinaintea primului câmp.

Afişarea câmpurilor memo se va face numai dacă în <lista-expr> figurează denumirea lor.

Exemplu:

```
use student          && presupunem un fişier cu date despre studenți
list                 && list are domeniul implicit ALL
```

#	cods	numes	grupa	anul	numec
1	1	Mihai Andrei	1	1	info
2	2	Boila Adela	1	1	mate
3	3	Albu Andrei	1	2	info
4	4	Fratean Doina	2	1	engleza
5	5	Suceava Dana	3	2	chineza

Exemplu:

```
USE mijloacef
DISPLAY FIELDS  cod, denumire, valoare
NOTE  se afiseaza doar campurile din lista
DISPLAY ALL FOR stare=.T.
NOTE se afiseaza doar mijloacele fixeîn folosinta
USE
```

Observație:

Afişarea afişării numelor de câmpuri în comenzile LIST/DISPLAY este reglementată de comanda SET HEADING ON/OFF. Implicit comanda este pe valoarea OFF.

Exemplu:

```
USE mijloacef
SET  HEADING ON
LIST              && se afiseaza cu antet
SET  HEADING OFF
LIST              && se afiseaza fara antet
USE
```

Căutarea inregistrarilor intr-o baza dedate

Poziționarea într-o bază de date se poate face şi prin căutarea unui anumit articol care îndeplineşte o condiție. Sunt mai multe posibilități de acces rapid la o bază de date ordonată (indexată) sau nu. Vom prezenta câteva comenzi şi funcții pentru baze neordonate.

Comanda de căutare secvențială LOCATE:

LOCATE FOR <COND> [<domeniu>]

Se caută primul articol care îndeplineşte condiția <cond> şi, dacă se găseşte indicatorul de înregistrare, se poziționează pe articolul respectiv.

Dacă nu se găseşte nici un articol care să verifice condiția, pointerul de fişier va indica sfârşitul fişierului sau domeniului indicat prin clauza <domeniu>.

Funcții de testare a succesului sau insuccesului căutării cu LOCATE sunt:

1. FOUND ()   && întoarce .T. dacă articolul a fost găsit
2. EOF ()   && întoarce .F. dacă articolul a fost găsit

Atenție! Nu se poate folosi funcția EOF() dacă <domeniu> este diferit de ALL.

Comanda de continuare a căutării următorului articol cu aceeaşi cheie este.

CONTINUE

Comanda găseşte următoarea înregistrare care respectă condiția specifică în ultima comandă LOCATE aplicată asupra bazei de date-active.

Funcția de căutare şi poziționare LOOKUP:

LOOKUP (<camp1), <exp>, <camp2> [,<expc>])

Funcția caută într-o bază de date prima apariție a unei expresii date. În caz de reuşită indicatorul de înregistrare se poziționează pe înregistrarea căutată, funcția returnând valoarea câmpului <câmp1>. Dacă articolul nu se găseşte, funcția returnează şirul vid; <câmp2> este câmpul a cărei valoare este cercetată, <exp> este expresia de căutat.
Exemplu: sa se gaseasca primele doua mijloace fixeîn folosinta din baza de date mijloacef:

```
CLOSE ALL
USE mijloacef
LOCATE  FOR   stare=.t.
? FOUND ( )
.T.
? EOF ( )
.F.
```

? RECNO ( )
1
CONTINUE
? FOUND ( )
.T.
? EOF ( )
.F.
? RECNO ( )
3
USE

Exemplu: sa se afiseze numele primului mijloc fix nefolosit din baza de date mijloacef.

CLOSE ALL
USE mijloacef
LIST
? LOOKUP (denumire, .F. , stare)
?REXNO ( )
??denumire
3 Autocamion M100
USE

Comanda de salt şi poziționare:

GO [TO] <n>/TOP/BOTTOM

Comanda GOTO <n> poziționează pe articolul cu numărul <n> în baza de date activă. GO TOP – pe primul articol în baza de date activă, GO BOTTOM – pe ultimul articol în baza de date activă.
Exemplu:

USE mijloacef
GOTO 2 && inregistrarea curenta va fi 2
? RECNO ( )
2
GO RECORD RECNO ( ) + 1
NOTE pozitionarea pe inregistrarea urmatoare (inregistrarea curenta + 1)
DISPLAY NEXT 1
GO TOP &&pozitionare pe inregistrarea 1

? RECNO ( )
1
GO BOTTOM && pozitionare pe ultima inregistrare
? RECNO ( )
4
USE

Urmatoarele instructiuni sunt echivalente (pentru o baza de date neindexata):

GO 1
GOTO 1
GO RECORD 1
GO TOP
GO TOPîn SELECT

Comanda de avans şi poziționare:

SKIP [+/-] <n>

Comanda face avansul (+) sau devansul (-) în baza de date peste <n> articole.
Observație: Deschiderea unei baze de date se face cu poziționarea pe primul articol.
Exemplu:

USE mijloacef
? RECNO ( )
1
SKIP 2
DISPLAY RECORD RECNO ( )
SKIP –1
?RECNO ( )
2
USE

Urmatoarele instructiuni sunt echivalente (pentru o baza de date neindexata):

SKIP
SKIP 1

SKIPîn SELECT ( )
GO TO RECORD RECNO ( ) + 1

Duplicarea unei baze de date

Una din operațiile frecvent folosite în aplicațiile economice este copierea întregului conținut al unei baze de date, sau o parte din el, în alt fişier. Cu această ocazie se creează o nouă bază de date care poate avea ca structură toate câmpurile din vechea bază de date, sau o parte din ele.

Copierea (duplicarea) unei baze de date se poate realiza prin comanda:

COPY TO <fis.dbf> [FIELDS <lista-câmp>] [<domeniu>] [FOR <conditie>][WHILE <conditie>]

Se vor copia articolele bazei de date active într-o nouă bază de date cu numele precizat în clauza TO <fis.dbf>. Clauza FIELDS enumeră câmpurile care vor forma structura noii baze de date. Clauzele de selecție <domeniu>, FOR, WHILE permit preluarea parțială a articolelor. Dacă există un fişier bază de date cu numele dat în clauza TO, se cere acordul de suprascriere.

Exemplu:
use mijloacef   && obținem exact aceleaşi date în două fişiere
copy to manevra
copy to manevra fields denumire for stare=.t.
&& se va crea o manevră cu mijloacele fixe existente

Actualizarea bazelor de date

Punerea la zi a bazelor de date se face prin diferite operații grupate de obicei în termenul de actualizare. Astfel, la o bază de date se pot adăuga articole noi, se pot insera articole înainte sau după o anumită înregistrare a bazei de date curente, se pot modifica valorile câmpurilor din baza de date, sau, în fine, se pot şterge logic sau fizic articole.

Adăugarea articolelor:

Completarea interactivă de date în continuarea celor existente într-o bază de date se poate face prin comanda APPEND pe care am discutat-o la operația de încărcare (văzută în termenii bazelor de date ca o adăugare pe o structură vidă).

O altă posibilitate de adăugare într-o bază de date este dată de comanda:

APPEND BLANK

La sfârşitul bazei de date activă se va adăuga un articol vid urmând ca ulterior acesta să fie completat cu valori potrivite. Reamintim că un câmp vid are una din valorile: zero pentru câmpul numeric, spațiu pentru câmpul caracter, .F. pentru câmpurile logice, valoarea {} sau {//} pentru dată calendaristică.

Adăugarea articolelor din altă bază de date se face prin comanda:

APPEND FROM <fis.dbf> [FOR <cond>]

Baza de date activă primeşte în continuarea articolelor sale înregistrările din altă bază de date specificată în clauza FROM. În mod implicit se preiau toate câmpurile. Condiția dată în clauza FOR este testată după plasarea articolului pe noua structură, aceasta fiind explicația pentru care expresia logică <cond> trebuie să conțină câmpuri ale bazei de date destinație.

Exemplu:

USE mijloacef
APPEND BLANK && adăugăm un articol vid

Inserarea articolelor:

De multe ori avem nevoie ca noile articole care trebuiesc trecute într-o bază de date să fie plasate fizic într-o anumită poziție, între articolele existente deja, sau înaintea primului, etc.

Comanda INSERT are urmîtoarea sintaxă:

INSERT [BLANK] [BEFORE]

Comanda INSERT asigură deschiderea ecranului de introducere a datelor şi permite operatorului trecerea directă a valorilor fiecărui câmp în parte (ca la comanda APPEND; se completează un singur articola care va fi plasat după articolul curent). Clauza BEFORE permite ca articolul introdus de operator să se aranjeze înaintea articolului curent. Clauza BLANK determină inserarea unui articol vid fără a deschide ecranul de introducere date.
Inserarea articolului vid se va face după (implicit) sau înaintea articolului curent (dacă este prezentă clauza BEFORE).

Exemplu:

USE mijloacef_n
GOTO 2 && se pozitioneaza indicatorul de inregistrari pe inregistrarea 2
INSERT BEFORE && se insereaza o noua inregistrareîn pozitia 2
USE

Modificarea bazelor de date

Activitatea de corectare a valorilor depuse în câmpuri la încărcarea bazei de date se poate face interactiv sau prin comenzi-program.

Interactiv, sistemul deschide un ecran de editare şi utilizatorul intervine direct asupra datelor necesare a fi modificate (EDIT, CHANGE, BROWSE).

Uneori căutarea directă a zonelor de corectat este inutilă sau prea obositoare sau vrem ca utilizatorul să nu „vadă" chiar tot; atunci se utilizează o comandă de corectare mai puternică REPLACE.

Comanda REPLACE are formatul general:

REPLACE <câmp1> WITH <exp1> [<câmp2> WITH <exp2>…]
[domeniu] [FOR<cond>][WHILE <cond>]

Comanda REPLACE permite înlocuirea valorii existente în câmpul <câmp1> cu valoarea expresiei <exp1>, a valorii existente în <câmp2> cu valoarea <exp2> ş.a.m.d.
Corecția vechilor valori se face pe domeniul indicat în clauza <domeniu>, pentru acele articole din domeniu care verifică condițiile din clauzele FOR şi WHILE (dacă există). Domeniul implicit este articolul curent.

Exemplu: la baza de date mijloacef se va adauga o noua inregistrare, cu urmatorul continut;

COD: vopsea
DENUMIRE: vopseaîn ulei, albastra
……..

secventa de comenzi care realizeaza acest lucru este:

USE mijloacef
APPEND BLANK
REPLACE cod WITH 'vopsea' denumire WITH ' vopseaîn ulei, albastra';
……….

LIST
USE

Ştergerea articolelor din baza de date

Punerea la zi a bazei de date este de nerealizat fără posibilitatea ştergerii articolelor care, fie au fost introduse eronat de operator, fie nu mai sunt valabile pentru colecția de date respectivă. De exemplu, într-o evidență BIBLIOTECA, necesară prelucrărilor curente relativ la cărți şi cititori, ce rost ar avea să reținem informații despre cărțile pierdute, sau scoase din uz, sau informații despre cititorii bibliotecii care şi-au retras legitimațiile de intrare şi nu mai au acces la bibliotecă?

Desigur, astfel de date sunt uneori interesante dar, să nu uităm că viteza de răspuns la interogări este direct dependentă de mărimea fişierului. Pentru situațiile în care, eventual, „cineva" ar cere date despre cărțile sau persoanele care au fost în evidențele bibliotecii şi nu mai sunt, se pot crea arhive (istorice), eliberând spațiul fişierelor de lucru.

Ştergerea articolelor se poate face logic, folosind o marcare vizibilă la afişare prin caracterul "*" înaintea primului câmp. Marcarea pentru ştergere poate fi anulată sau, dacă s-a considerat oportună ştergerea articolului, atunci acesta se poate şterge fizic din baza de date.

Comanda de **marcare pentru ştergere** este DELETE:

DELETE[<domeniu>][FOR<cond>] [WHILE<cond>]

Comanda marchează pentru ştergere articolele din domeniul precizat în clauza <domeniu> care îndeplinesc condițiile puse în clauzele FOR şi WHILE.
Implicit comanda acționează pe articolul curent.

Exemplu :

USE mijloacef
CLEAR
SET DELETED OFF

DELETE FOR MOD (RECNO ( ), 2) = 0
NOTE se şterg inregistrarile cu număr de ordine par
LIST
NOTE toate inregistrările din baza de date sunt afişate, cele şterse avand un asterisc;
în dreptul lor
GO TO 2
DISPLAY && ănregistrarea este afişata chiar dacă este marcată pentru ştergere;
Display având ca domeniu implicit inregistrarea curentă
USE

Observație: Starea de articol marcat pentru ştergere nu influențează în mod obişnuit nici comanda de afişare (observăm „*" înaintea primului câmp!), nici o eventuală căutare prin LOCATE, o copiere (COPY), o sortare (SORT), etc.
Acest lucru se datorează valorii OFF pe care este poziționată implicit comanda comutator SET DELETED ON/OFF.
Valoarea ON determină ignorarea articolelor marcate pentru ştergere.

Ştergerea fizică a articolelor se poate face prin două comenzi: PACK şi ZAP.

PACK ZAP

Comanda PACK permite ştergerea fizică din fişier a tuturor articolelor marcate anterior. Nu mai este nici o posibilitate de recuperare a acestor date.
Exemplu: se sterge inregistrarea a 5-a din baza de date mijloacef.
USE mijloacef
LIST
DELETE RECORD 5 && se marchează pentru ştergere a 5-a inregistrare
PACK && se şterge fizic a 5-a inregistrare
LIST
USE

Comanda ZAP permite ştergerea definitivă din fişier a tuturor articolelor, fără ca în prealabil să fi avut loc o operație de marcare. Este similară secvenței de comenzi:

delete all
pack

***Observație*: **Trebuie să fim atenți la poziționarea comutatorului SET SAFETY <u>ON</u>/OFF care, pe valoarea ON cere acordul la ştergere.

Anularea marcajelor de ştergere se face prin comanda RECALL:

RECALL [<domeniu>] [FOR <cond>] [WHILE<cond>]

Comanda RECALL permite revenirea unui articol la starea anterioară operației de ştergere numai dacă ştergerea a fost logică (prin comanda DELETE).

Acțiunea comenzii are ca domeniu implicit articolul curent. Prin specificarea clauzelor [<domeniu>], FOR şi WHILE putem selecta articolele şterse prin comanda DELETE, a căror marcaje dorim să le anulăm.

Exemplu:

USE mijloacef
DELETE FOR RECNO ( ) <=3
&& se şterg primele 3 inregistrări
LIST && se observă efectul ştergerii
RECALL ALL && sunt refăcute toate înregistrările. Cele care nu erau;
marcate pentru ştergere nu sunt afectate
LIST
USE

Actualizarea interactivă a bazelor de date

Actualizarea datelor din bazele de date se poate face direct de către utilizator prin intermediul ecranelor de actualizare EDIT şi BROWSE, deschise la comenzile cu acelaşi nume. Prin intermediul acestor ecrane se pot vizualiza, introduce, edita şi şterge date. Comanda EDIT are formatul general:

EDIT / CHANGE [NOINIT] [NOAPPEND] [NOMENU] [NOEDIT]
[NODELETE] <nr-articol>[FIELDS <lista-câmp>]
[<domeniu>] [FOR <cond>] [WHILE <cond>]

Comanda deschide ecranul EDIT afişând înregistrarea cu numărul <nr-articol> a bazei de date active. Structura va cuprinde toate câmpurile – dacă lipseşte clauza FIELDS – sau câmpurile enumerate în <lista-câmp>. După efectuarea corecțiilor asupra articolului curent, se trece automat la următorul articol. Se poate ieşi cu salvare <ctrl>w<> sau cu abandonare <ctrl><q>.

Clauza FIELDS permite limitarea editării numai la câmpurile enumerate; în lipsă se vor afişa şi manevra toate câmpurile din structură.

Clauza NOINIT permite folosirea clauzelor unei comenzi EDIT anterioară fără a le mai specifica în comanda curentă.

Clauzele care urmează sunt folosite în program în vederea limitării acțiunii unui utilizator oarecare la o bază de date. Astfel, clauza NOAPPEND interzice adăugarea de noi articole în fişier; în lipsa clauzei, acest lucru este posibil. NOMENU nu afişează linia de meniuri şi împiedică accesul la meniuri. NODELETE împiedică ştergerea accidentală de articole. Cu NOEDIT articolele sunt doar afişate, se interzice editarea lor.

Exemplul : având baza de date mijloacef, vom deschide o ferestră de editare în care vom afişa doar câmpurile COD, VECHIME, STARE. Primul şi ultimul sunt câmpuri simple ale bazei de date, VECHIME

fiind un câmp calculat în funcție de data instalării (câmpul DATA_INST din baza de date) şi data curentă.

Exemplul:

```
                CLOSE ALL
                USE mijloacef
                CHANGE FIELDS cod:R, stare
                NOTE se afiseaza câmpurile cod si stare, dar se
poate modifica doar câmpul stare
                USE
```

Comanda BROWSE afişează articolele din baza de date sub forma unui tabel:

BROWSEE [FIELDS<câmp1> [/R][/<dim>] [/câmp-calc1>=<exp1>]
[, <câmp2> [/R][/<dim>] [/<câmp-calc2=<exp2>],..]
[LOCK <nr>] [WIDTH <exp2>] [FREEZE <nume-câmp>]
[NOINIT][NOAPPEND][NOMENU][NOEDIT]
[NODELETE]

Comanda BROWSE permite deschiderea unui ecran special, numit ecran BROWSE, prin intermediul căruia se pot actualiza baza de date, cu toate operațiile ce țin de acest lucru.
Clauza FIELDS permite enumerarea câmpurilor care vor forma coloanele tabelului; în lipsa clauzei se rețin toate câmpurile din baza de date, în ordinea structurii. Pentru un câmp putem interzice editarea: /R, putem preciza dimensiunile la afişare /<dim>. În lista de câmpuri pot apare şi câmpuri calculate care primesc un nume şi o expresie de calculat. Câmpurile calculate nu sunt editate ci numai afişate; dar valorile din acestea se modifică odată cu modificările în câmpurile care conțin expresia de calculat.

Clauza LOCK <nr> permite înghețarea pe ecran a primelor <nr> coloane (câmpuri) în timpul defilării tabloului BROWSE spre stânga sau spre dreapta.

Se recomandă ca în structura conceptuală să se aranjeze la început informațiile de identificare a unui obiect. Dacă acest lucru nu este realizat, putem schimba ordinea de afişare pe ecran a câmpurilor

cu clauza FIELDS, astfel încât să avem pe primele coloane succesive, informațiile necesare.

Clauza WIDTH <lung> dă posibilitatea programatorului să fixeze dimensiunea maximă a coloanelor. Desigur, dacă lungimea câmpurilor este mai mică decât dimensiunea indicată în clauza WIDTH, aceasta din urmă este ignorată.

Clauza FREEZE <nume-câmp> permite menținerea cursorului pe o singură coloană.

Clauzele celelalte sunt identice cu cele ale comenzii EDIT/CHANGE. Meniul ferestrei BROWSE este prezentat în caietul de laborator.

Observații: Sunt câteva particularități FoxPro legate de actualizare:

1. Comanda APPEND FROM are clauza FIELDS <lista-câmpuri> prin care se poate indica lista de câmpuri în care se vor introduce valori din fişierul sursă:

APPEND FROM <fis.dbf> [FOR <cond>] [FIELDS <lista- câmp>]

2. Comanda PACK are două clauze în plus [MEMO] [DBF]. Clauza MEMO se foloseşte atunci când se doreşte diminuarea spațiului disc nefolosit din fişierul memo asociat, fără a afecta fişierul bazei de date. Clauza DBF se foloseşte pentru a şterge înregistrările marcate pentru ştergere din baza de date, fără a modifica fişierul memo.

Lucrul cu câmpurile memo

Sunt situații când, într-un anumit câmp al unei baze de date, trebuiesc memorate *cantități variabile* de informații. Astfel, într-o evidență a personalului unei unități, informația studii poate avea lungimea variind de la câteva caractere („Liceu Informatică, Iaşi") la câteva zeci de caractere („Academia de Studii Economice, Facultatea de Cibernetică Economică, Bucureşti"). Care va fi dimensiunea câmpului (de tip caracter) care va reține această informație?

Desigur, câmpul va fi dimensionat maxim (să zicem 100 caractere) pentru a putea reține corect toate informațiile, dar să nu neglijăm faptul că la o parte din articole, poate chiar la majoritatea, vor rămâne zone nefolosite. Pe de altă parte, este ştiut faptul că lungimea articolului determină direct mărimea timpului de răspuns la interogările utilizatorilor. Deci, informația este în întregime memorată, dar în defavoarea timpului de acces şi a utilizării spațiului pe disc.

Ce se întâmplă dacă, *în majoritatea aplicațiilor nu avem nevoie de aceste informații* despre studiile personalului?

O soluție ar fi memorarea studiilor într-o bază de date auxiliară, legată de fişierul principal prin intermediul unui cod. O altă soluție este folosirea tipului memo.

O bază de date, care conține cel puțin un câmp memo, are asociat un fişier suplimentar în care sunt depuse informațiile propriu-zise. Fişierul asociat poartă acelaşi nume ca baza de date şi se deschide simultan cu aceasta.

Accesul sistemului la conținutul unui câmp memo al unui anumit articol se face în modul următor:

- se selectează articolul şi câmpul memo dorit;
- se citeşte adresa înregistrată în câmpul memo;
- se localizează zona din fişierul memo care corespunde adresei citite;
- se permite accesul la informația de la această adresă.

Încărcarea unor date într-un câmp memo, la o anumită înregistrare, se face după următoarele etape:

- se găseşte un spațiu liber în fişierul memo asociat bazei de date, suficient pentru memorarea tuturor datelor şi se încarcă aceste date în spațiul respectiv;
- se completează la înregistrarea dorită, în câmpul memo, adresa zonei din fişierul memo unde s-au încărcat datele.

Încărcarea unor date într-un câmp memo se poate face fie direct de către utilizator, caracter cu caracter, într-o fereastră de editare, fie prin citirea acestor caractere dintr-un fişier text.

Introducerea directă de către utilizator se face prin intermediul comenzilor de actualizare APPEND, EDIT, BROWSE. După poziționarea pe articolul al cărui câmp memo dorim să-l introducem sau să-l edităm, vom deschide fereastra de editare prin tastele <ctrl><home>. Ieşirea cu salvare se face prin tastele <ctrl><end> / <ctrl><w>.

Pentru a edita un câmp memo, fără a mai trece prin fereastra de editare Change, se poate folosi comanda MODIFY MEMO, care deschide direct o fereastră de editare pentru câmpul memo specificat, al înregistrării curente din baza de date activă.

MODIFY MEMO <câmp memo 1>[,<câmp memo 2>.,]
[NOEDIT]
[NOWAIT][RANGE <expN1>, <expN2>]
[WINDOW<nume fereastra>]

Clauza NOEDIT nu permite modificarea conținutului câmpului memo.
Clauza NOWAIT se foloseşte numai în interiorul unui program şi are ca efect continuarea execuției programului, după deschiderea ferestrei de editare, fără a mai aştepta ca utilizatorul să modifice câmpul memo respectiv.
Clauza RANGE se foloseşte atunci când se doreşte ca numai o porțiune din câmpul memo să fie editată, şi anume partea cuprinsă între <expN1> şi <expN2>.
Clauza WINDOW permite deschiderea ecranului de editare într-o fereastră.
Exemplu:

USE mijloacef
GO TO 2
MODIFY MEMO loc_folos RANGE 1,40
NOTE se modifică înregistrarea 2, câmpul loc_folos, numai primele 40 de caractere.
GO TO 4
MODIFY MEMO loc_folos NOEDIT
NOTE se afişează câmpul loc_folos al înregistrării 4 fără a se permite modificarea sa.
USE

Închiderea ferestrei memo de editare se poate face folosind comanda CLOSE MEMO.

CLOSE MEMO <câmp memo> [, <câmp memo2>.....] | ALL

Se vor închide ferestrele de editare corespunzătoare câmpurilor memo specificate în listă, salvându-se eventualele modificări făcute acestor câmpuri. La închiderea unei baze de date se vor închide, de asemenea, şi ferestrele memo deschise în acel moment, pentru câmpurile memo ale bazei de date. Clauza ALL are ca efect închiderea tuturor ferestrelor memo deschise, pentru toate zonele de lucru.

Exemplu:

USE mijloacef
MODIFY MEMO loc_folos NOWAIT
NOTE se deschide fereastra de editare a câmpului memo loc_folos;
Fără a aştepta modificarea câmpului
WAIT 'Aşteptați TIMEOUT 5
NOTE se face o pauză de 5 secunde
CLOSE MEMO loc_folos
NOTE se închide fereastra memo respectivă
USE

Introducerea datelor dintr-un fişier text se face prin comanda:

APPEND MEMO <câmp memo> FROM <fisier> [OVERWRITE]

Întregul conținut al fişierului se adaugă sau se suprascrie (clauza OVERWRITE) în câmpul memo.

Exemplu: se va adăuga o nouă înregistrare la baza de date mijloacef, câmpul LOC_FOLOS fiind completat din fişierul ADRESA.TXT ce conține informațiile respective.

USE mijloacef

```
APPEND BLANK
REPLACE cod WITH ' Maşina '
REPLACE denimire WITH ' DACIA 1410 break'
REPLACE data_inst WITH ( 08 /21/99)
REPLACE stare WITH .T.
APPEND MEMO loc_folos FROM adresa.txt OVERWRITE
```

Operațiunea inversă, de extragere dintr-un câmp memo a informațiilor într-un fişier text peste, sau în continuarea vechiului conținut, se face prin comanda:

COPY MEMO <câmp memo> to <fişier> [ADDITIVE]

Exemplu:

```
USE mijloacef
COPY MEMO loc_folos TO adresa.txt
NOTE se copiază prima adresă
GO TO 2
COPY MEMO loc_folos TO adresa.txt ADDITIVE
NOTE se copiază a doua adresă
GO TO 3
COPY MEMO loc_folos TO adresa.txt ADDITIVE
NOTE se copiază a treia adresă
MODIFY FILE adresa.txt NOEDIT
NOTE se vizualizează fişierul adresa.txt
USE
```

Exemplu: să considerăm baza de date mijloacef în care se doreşte schimbarea între ele a locurilor de folosință ale înregistrărilor 2 şi 4. Aceasta se va realiza folosind ca intermadiar fişierul ADRESA.TXT.

```
CLOSE ALL
USE mijloacef
USE mijloacefîn 2 AGAIN
NOTE se deschide baza de date mijloacef şi în zona de lucru
2
GO TO 2
COPY MEMO loc_folos TO adresa.txt
```

NOTE se copiază câmpul memo loc_folos al înregistrării 2 în fişierul adresa.txt
GO TO 4 în B
REPLACE loc_folos WITH b. loc_folos
NOTE se înlocuieşte câmpul memo loc_folos al înregistrării 2 (zona de lucru 1);
cu câmpul memo loc_folos al înregistrării 4 (zona de lucru 2)
GO TO 4
APPEND MEMO loc_folos FROM adresa.txt OVERWRITE
NOTE se copiază conținutul fişierului adresa.txt în câmpul loc_folos al înregistrării 4;
(zona de lucru 1)
BROWSE
CLOSE ALL

O altă modalitate de a schimba cele două câmpuri memo din baza de date este reprezentată de următorul program:

```
CLOSE ALL
USE mijloacef
USE mijloacefîn 2 AGAIN
GO TO 2
V=  loc_folos
GO TO  4 în  B
REPLACE loc_folos WITH b. loc_folos
GO TO 4
REPLACE loc_folos WITH v
BROWSE
CLOSE ALL
```

Funcții relative la câmpurile memo:
1. MLINE(<câmp memo>, <numar>) extrage din câmpul memo linia dată prin numărul ei.
2. MEMLINES (<câmp memo>) dă numărul de linii al câmpului memo.

Observații: Ambele funcții sunt influențate de comanda:

Exemplu: să presupunem că în câmpul loc_folos al înregistrării curente din baza de date mijloacef avem următorul text:

Bucureşti
Strada Cuza Vodă, nr. 54
Sector 4

Atunci:

? MEMLINE (loc_folos)
NOTE numărul de linii din câmpul memo
3
? MLINE (loc_folos, 2)
Strada Cuza Vodă, nr. 54
? MLINE (loc_folos, 1, 0)
Bucureşti
? MLINE (loc_folos, 1, 1)
ucureşti
? MLINE (loc_folos, 1, 11)
Strada Cuza Vodă, nr. 54
? MLINE (loc_folos, 1, 12)
trada Cuza Vodă, nr. 54
? MLINE (loc_folos, 4) ==" "
.T.

Valorile returnate de funcțiile MEMLINE ( ) şi MLINE ( ) sunt influențate de comanda:

SET MEMOWIDTH TO <număr>

Comanda fixează lungimea liniilor din câmpul memo la <număr>. Implicit lungimea maximă este de 50 caractere; minim pot fi 8 caractere.

Exemplu: vom folosi acelaşi câmp memo ca şi la exemplul anterior:

SET MEMOWIDTH TO 60
? loc_folos
Bucureşti
Strada Cuza Vodă, nr. 54
Sector 4

```
? MEMLINE (loc_folos )
3
? MLINE  (loc_folos, 2 )
Strada Cuza Vodă, nr. 54
SET MEMOWIDTH TO 12
? loc_folos
Bucureşti
Strada Cuz
a Vodă, nr.
54
Sector 4
? MEMLINE (loc_folos )
5
? MLINE  (loc_folos, 2 )
Strada Cuz
```

Observaţii: există o comandă FoxPro prin care se poate deschide o fereastră de editare pentru fiecare câmp memo:

Transfer de date între fişiere şi tablouri

De multe ori folosirea tablourilor de date este preferabilă folosirii fişierelor de date, pentru că viteza de accesare a memoriei este net superioară vitezei de accesare a discului. De exemplu, sortarea unui masiv este mult mai rapidă decât sortarea unui fişier. Comunicarea între baze de date şi tablouri se realizează în ambele sensuri prin comenzi corespunzătoare.

Adăugarea datelor dintr-un tablou într-o bază de date se poate face prin comanda:

APPEND FROM ARRAY <tablou> [FOR<cond>]

Se adaugă la baza de date articolele preluate dintr-un tablou; fiecare linie corespunde unei înregistrări; coloanele se copiază în ordinea câmpurilor; se ignoră elementele în plus; câmpurile suplimentare se completează automat cu valori vide. Se face

conversia la tipul câmpului, iar dacă lungimea este mai mică pentru câmpuri caracter, se înlocuieşte cu asterisc.

Trecerea articolelor din baza de date într-un tablou se face cu ajutorul comenzii:

COPY TO ARRAY <tablou> [FIELDS <lista câmpuri>] [<domeniu>] [FOR<cond>] [WHILE<cond>]

Fiecare articol devine o linie. Dacă numărul de înregistrări depăşeşte numărul de linii, se ignoră cele în plus. Spre deosebire de APPEND, se pot selecta câmpurile care vor fi copiate în tablou prin clauza FIELDS <lista câmpuri>.

Corecția unor valori ale bazei de date prin utilizarea tablourilor se face la comanda:

REPLACE FROM ARRAY <tablou> [FIELDS<lista câmpuri>] [<domeniu>][FOR<cond>] [WHILE<cond>]

Se înlocuiesc cu elementele din tabloul specificat acele valori care corespund câmpurilor din clauza FIELDS (implicit toate câmpurile) pentru articolele selectate prin clauzele de selecție: <domeniu>, FOR, WHILE. Comanda REPLACE are ca domeniu implicit articolul curent.

Observații: Sunt câteva comenzi noi în FoxPro relativ la comunicarea între tablouri şi baze de date: GATHER, SCATTER:

a.Adăugarea datelor dintr-un tablou în baza de date se face prin:

GATHER FROM <tablou>/MEMVAR [FIELDS<lista-câmp>][MEMO]

Comanda permite copierea conținutului tabloului <tablou> sau a unei variabile de memorie în baza de date activă, în înregistrarea curentă, în câmpurile precizate în clauza FIELDS.
MEMVAR desemnează variabilele de memorie folosite având acelaşi nume cu câmpurile în care se vor copia.
MEMO este necesară dacă printre câmpurile de copiat se găseşte şi un câmp memo.

b. Copierea datelor din baza de date într-un tablou se face prin comanda:

SCATTER [FIELDS <lista-câmp>][memo] TO <tablou>/MEMVAR

Câmpurile din baza de date activă specificate în clauza FIELDS (sau toate câmpurile, opțiune implicită) se vor copia în tabloul din clauza TO <tablou> sau în variabilele speciale (clauza MEMVAR).

Funcții relative în fişiere

nrc	funcția	explicații
1	FIELD (<expn>) [,<alias>]	dă numele câmpului cu număr de ordine <expN>
2	RECSIZE () [<alias>]	întoarce dimensiunea în octeți a structurii bazei
3	TYPE (<câmp>)	întoarce tipul unui câmp precizat ca şir de caractere
4	FLDCOUNT ()	întoarce numărul de câmpuri din structura fişierului
5	RECHO() [<alias>]	întoarce numărul articolului curent
6	EOF() [<alias>]	întoarce .T. dacă s-a ajuns la sfârşitul de fişier
7	BOF() [<alias>]	întoarce .T. dacă s-a ajuns înaintea primului articol
8	RECCOUNT() [<alias>]	întoarce numărul de articole din baza de date activă
9	FOUND()	întoarce .T. dacă articolul căutat a fost

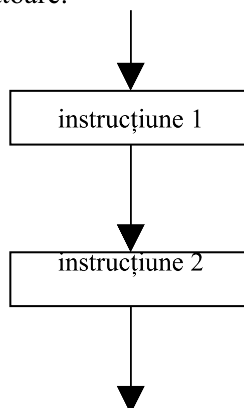
		găsit
10	SELECT()	dă prima zonă liberă (ca număr)
11	MLINE(<câmp>, <numar>)	extrage din câmpul memo linia dată prin numărul ei
12	MEMLINES (<câmp>)	dă numărul de linii al câmpului memo
13	LOOKUP (<câmp1>, <exp>, <câmp2>)	întoarce valoarea <câmp1> a articolului care verifică <exp> în câmpul <câmp2>

Programare structurată

Programarea structurată este o metodă independentă de limbajul de programare care impune reguli stricte în conceperea algoritmilor de rezolvare a problemelor. Ea disciplinează realizarea algoritmilor prin restrângerea structurilor de control utilizabile la un număr redus de tipuri. Acest principiu de bază derivă din teorema de structură a lui Böhm-Jacobini care arată că orice algoritm se poate construi folosind doar trei tipuri de structuri de control: secvențială, alternativă, repetitivă. Limbajul dBASE (ca şi FoxPro) permite descrierea acestor structuri prin comenzi corespunzătoare:

Structuri de control de bază

Structura secvențială – controlul este transferat de la o operație la alta în secvență.



Structura alternativă - permite alegerea între două alternative.

În unele situații algoritmul de rezolvare a problemei impune ramificarea programului în două ramuri condiționată de valoarea de adevăr a unei condiții , astfel încât, la executarea acestuia , în funcție

de datele problemei, calculatorul să poată executa una din cele două acțiuni posibile.

Instrucțiunea IF

Această facilitate este asigurată de instrucțiunea IF. Forma generală (două variante) a acestei instrucțiuni este:

1.Formatul pentru structura alternativă cu două ramuri

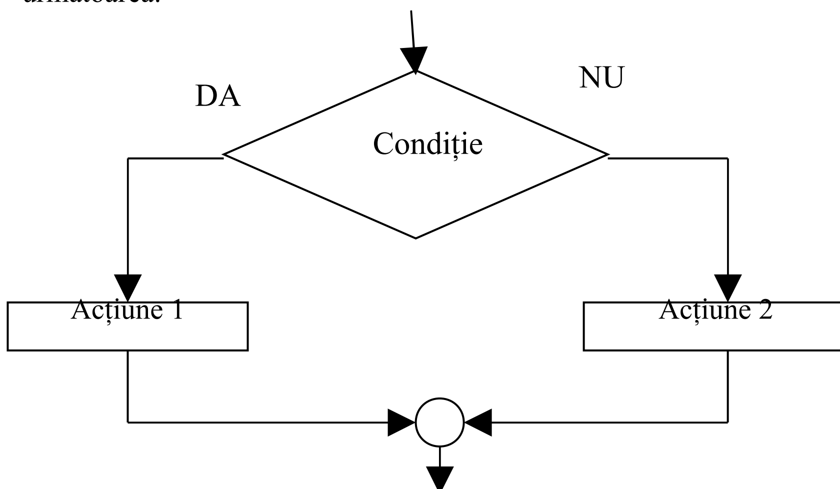
IF condiție
< instrucțiuni1>
ELSE
<instrucțiuni2>
ENDIF

2.Formatul pentru structura alternativă cu o ramură vidă

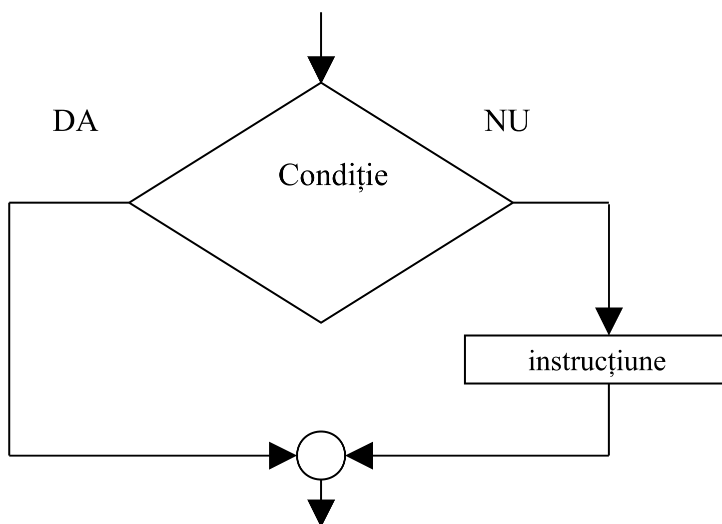
IF condiție
< instrucțiuni>
ENDIF

În limba engleză IF înseamnă „dacă” , iar ELSE înseamnă „altfel”. Funcționarea instrucțiunii IF poate fi explicată prin următoarele diagrame.

1.Formatul pentru structura alternativă cu două ramuri este următoarea:



2.Formatul pentru structura alternativă cu o ramură vidă



La întâlnirea unei instrucțiuni IF se va evalua în primul rând valoarea de adevăr a expresiei booleene condiție. În cazul în care aceasta are valoarea TRUE se va executa grupul de instrucțiuni <instrucțiuni1>, după care execuția comenzii se va termina.

Dacă condiția are valoarea FALSE şi dacă există ELSE se va executa instrucțiunile din grupul <instrucțiuni2>, după care execuția comenzii se va termina.

În ambele cazuri următoarea instrucțiune care se va executa este cea de după terminarea instrucțiunii.

Exemplu:

SET TALK OFF
CLEAR
a = 0
b = 0
@4,10 SAY ' Primul numar' GET a PICTURE '9999'
@5,10 SAY ' Al doilea numar' GET b PICTURE '9999'
READ
IF a>b
? " Primul număr este mai mare "   && se execută când a>b
ELSE

```
        ? " Al doilea număr este mai mare sau sunt egale"
        NOTE se execută când a<=b
ENDIF
?
IF b<> 0
        NOTE se execută când b este diferit de 0
        ? a, ":",b,"=",a/b
ENDIF
```

Un efect asemănător, de selecție dintre două variante, se obține prin funcția IIF () având sintaxa:

IIF (<expL>, <expr1>, <expr2>)

Această funcție evaluează expresia logică <expl> şi în funcție de rezultatul obținut, returnează valoarea uneia dintre expresiile <expr1> şi <expr2>:

- returnează valoarea obținută prin evaluarea expresiei <expr1> dacă <expl> este evaluată la .T.
- returnează valoarea obținută prin evaluarea expresiei <expr2> dacă <expl> este evaluată la .F.

Cele două expresii <expr1> şi <expr2> nu trebuie neapărat să aibă acelaşi tip, acestea putând fi de tip şir de caractere, dată calendaristică, logic sau numeric.

Exemplu:

```
 SET TALK OFF
CLEAR
a = 0
b = 0
@4,10 SAY ' Primul numar' GET a PICTURE '9999'
@5,10 SAY ' Al doilea numar' GET b PICTURE '9999'
READ
sir1=" Primul număr este mai mare "
sir2= " Al doilea număr este mai mare sau sunt egale"
? IIF a>b, sir1, sir2
```

Mai multe comenzi IF …..ENDIF pot fi imbricate, incluse una în alta, obținându-se condiționarea unui grup de instrucțiuni prin mai multe expresii logice, sau obținându-se selecții diverse între mai multe grupuri de instrucțiuni.

Exemplu:

```
SET TALK OFF
CLEAR
a = 0
b = 0
@4,10 SAY ‘ Primul numar’ GET a PICTURE ‘9999’
@5,10 SAY ‘ Al doilea numar’ GET b PICTURE ‘9999’
READ
sir1=” Primul număr este mai mare “
sir2= “ Al doilea număr este mai mare sau sunt egale”
sir3= “ Numerele sunt egale”
IF a>b
    ? sir1     && a>b
ELSE
    IF a<b
        ? sir2   && a<b
    ELSE
        ? sir3   && a=b
    ENDIF
ENDIF
```

În acest exemplu structura de comenzi IF….ENDIF poate fi înlocuită cu:

```
? IIF ( a>b, sir1, IIF ( a<b, sir2 , sir3))
```

în care avem două apeluri ale funcției Iif ( )incluse unul în altul.

Instrucțiunea CASE

Instrucțiunea IF permite selectarea uneia dintre cele două acțiuni în concordanță cu valoarea unei expresii booleene.

Instrucțiunea CASE este o generalizare a instrucțiunii IF în sensul că asigură executarea condiționată a uneia din cele câteva

acțiuni posibile, în funcție de valoarea unei expresii de tip scalar sau subdomeniu (numită selector), ca urmare permite alegerea din atâtea instrucțiuni câte valori distincte are tipul expresiei.

Sintaxa este:

DO CASE
CASE <expL1>
< instrucțiuni1>
[CASE <expL2>
< instrucțiuni2>
……………………..
CASE <expLN>
< instrucțiuniN>]
[OTHERWISE
< instrucțiuni>]
ENDCASE

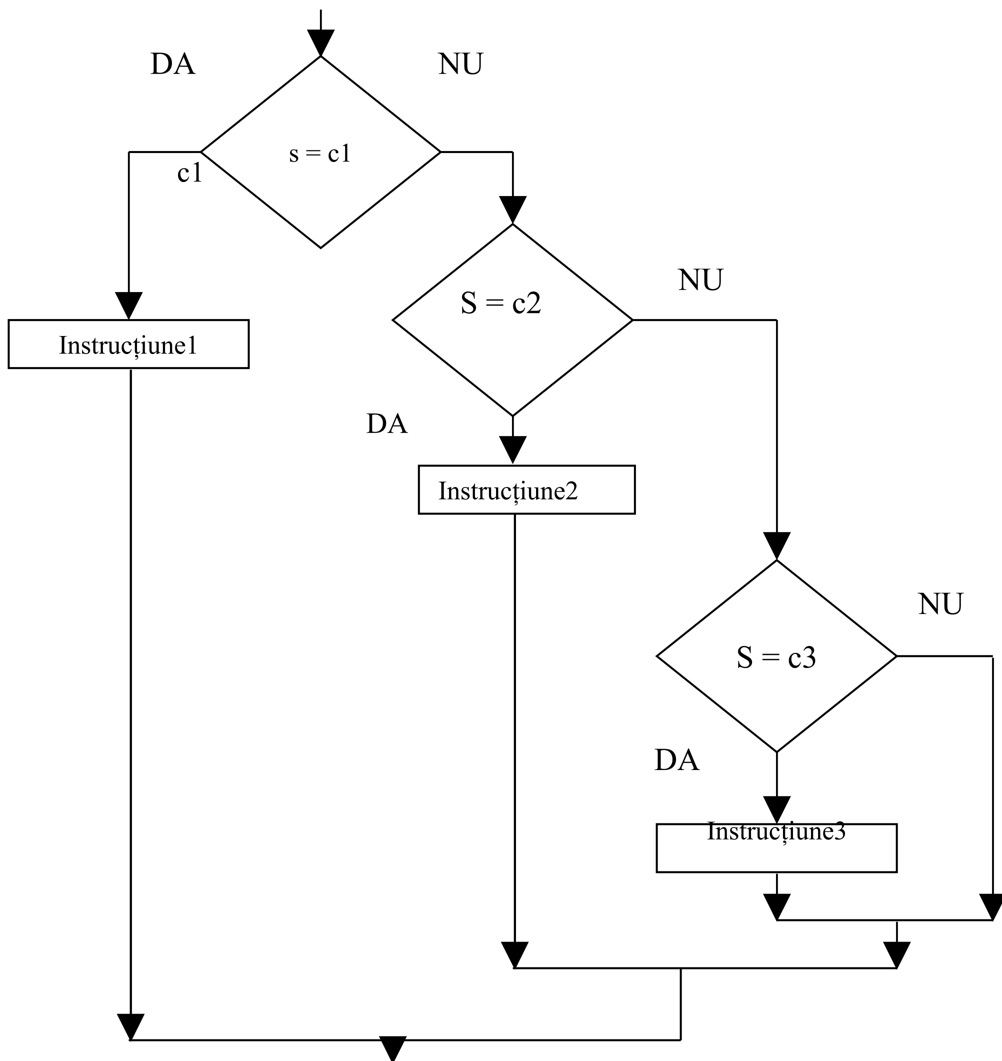
Comanda va determina execuția grupului de instrucțiuni pentru care expresia logică corespunzătoare are valoarea .T.. Execuția comenzii va decurge în modul următor:

Se evaluează prima expresie logică <expL1> şi dacă valoarea obținută este .T. se execută grupul de instrucțiuni <instrucțiuni1>. Dacă expresia logică <expL1> are valoarea .F. se trece la evaluarea următoarei expresii logice. După găsirea primei expresii logice cu valoarea .T. şi executarea grupului de instrucțiuni corespunzător, execuția comenzii se încheie, programul continuând cu prima comandă de după ENDCASE.

Dacă nici una dintre expresiile <expL1>, <expL2>…..<expLN> nu are valoarea .T., apar două cazuri:

- când nu există clauza OTHERWISE, execuția comenzii se încheie;
- în prezența clauzei OTHERWISE, se va executa grupul de instrucțiuni <instrucțiuni>, după care se trece la prima comandă de după ENDCASE.

Observație: numai unul dintre grupurile de instrucțiuni <instrucțiuni1> ……<instrucțiuniN> şi <instrucțiuni> va fi executat şi anume acela pentru care expresia logică este .T.



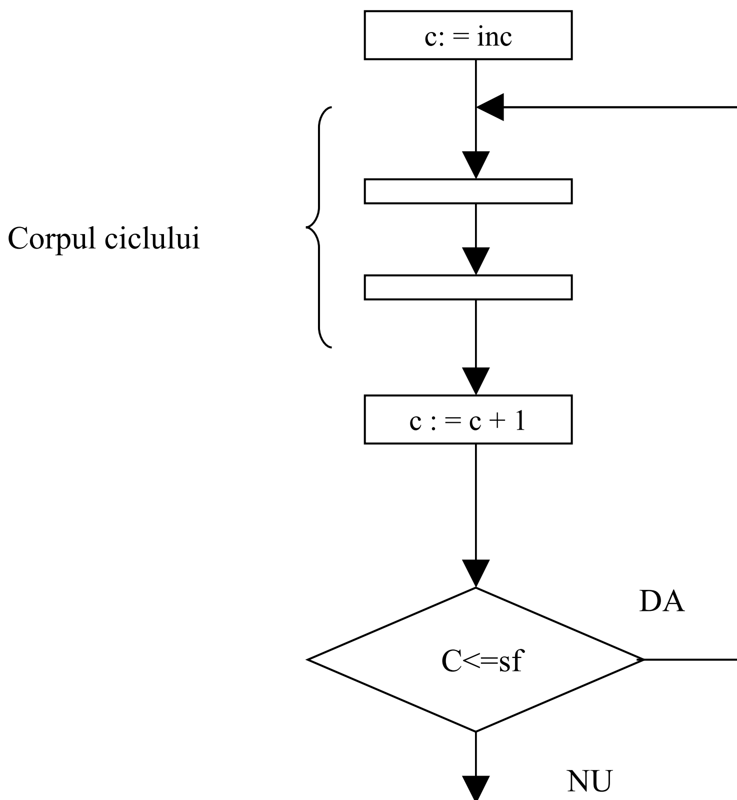
Exemplu:

SET TALK OFF

```
CLEAR
DIMENSION a (4 )
a (1) = ‘Primăvara’
a (2) = ‘Vara’
a (3) = ‘Toamna’
a (4) = ‘Iarna’
anotimp =’ Prim[vara’
 @ 6,26  GET anotimp FROM a FUNCTION ’&’
READ
@ 15,10 SAY  ’ Acest anotimp con’ine lunile: ’
DO CASE
        CASE anotimp = ’ Primăvara’
          ?? ’ Martie, Aprilie, Mai ’
        CASE anotimp = ’Vara’
          ??  ’ Iunie, Iulie, August ’
        CASE anotimp = ’Toamna’
          ?? ’ Septembrie, Octombrie, Noiembrie ’
        OTHERWISE
          ?? ’ Decembrie, Ianuarie, Februarie ’
ENDCASE
```

Structura repetitivă

a) Structura clclică – aceasta foloseşte un contor de ciclu care va determina numărul de repetări al executării acțiunilor din cadrul corpului ciclului (grup de acțiuni care se pot repeta de mai multe ori).



Unde:

c –contor de ciclu

inc – expresie

sf – expresie

Instrucțiunea FOR

In unele situații trebuie ca un grup de instrucțiuni să fie repetat de un anumit număr de ori bine precizat şi dinainte cunoscut. În această situație se utilizează instrucțiunea FOR.

Sintaxa instrucțiunii este:

a) Instrucțiunea FOR pentru valori crescătoare ale variabilei contor:

FOR <var> = <expN1> TO <expN2> [STEP <expN3>]
<instrucțiuni>
[EXIT]
[LOOP]
ENDFOR

La utilizarea instrucțiunii FOR trebuie să se țină seama de următoarele observații:

1. Nu este indicat modificarea variabilei contor în cadrul ciclului.
2. La ieşirea din ciclu, variabila contor va avea valoarea sfârşit (în cazul în care ieşirea din ciclu s-a făcut în pasul normal).
3. Ieşirea forțată din ciclul FOR se poate realiza cu instrucțiunea EXIT (prezentă în corpul ciclului). În acest caz, contorul va avea valoarea cu care a fost surprins de instrucțiunea EXIT.
4. Dacă început > sfârşit (sau sfârşit < început) este fals, se execută instrucțiunea imediat următoare după FOR.

Instrucțiunea FOR funcționează după următorul algoritm:

Valoarea inițială a variabilei contor va fi dată de evaluarea expresiei <expN1>. După fiecare execuție a grupului de instrucțiuni <instrucşiuni> variabila va fi incrementată sau decrementată cu o valoare constantă, dată de evaluarea expresiei <expN3>, dacă este prezentă clauza STEP, sau 1 când STEP lipseşte (absența clauzei STEP este echivalentă cu construcția STEP 1)

Când valoarea variabilei <var> creşte peste valoarea expresiei <expN2> (strict mai mare), în cazul unei valori pozitive a lui <expN3>, sau scade sub valoarea expresiei <expN2>, pentru o valoare negativă a expresiei <expN3>, se va ieşi din buclă, programul continuând cu următoarea comandă de după ENDFOR.

Exemplu: comanda

FOR i = 1 TO 4

```
            ? i
        ENDFOR
```

Este echivalentă cu grupul de comenzi:

```
        ? 1
        ? 2
        ? 3
        ? 4
```

În exemplul de mai sus i este variabila contor, 1 este valoarea inițială a variabilei, iar 4 este valoarea finală a acesteia. La fiecare execuție a lui ENDFOR variabila i va fi incrementată cu 1.

La ultima execuție a comenzii ? i valoarea lui i va fi 4. execuția lui ENDFOR are ca efect creşterea lui i cu 1, deci valoarea lui i va fi 5, după care depăşeşte valoarea finală 4. programul va continua cu prima instrucțiune de după ENDFOR.

Comanda:

```
        FOR i=4 to 1 STEP -2
            ? i
        ENDFOR
```

Este echivalentă cu :

```
        ? 4
        ? 2
```

Atenție! : cele trei expresii numerice, <expN1>, <expN2>, <expN3> sunt evaluate doar la intrarea în bucla FOR, modificarea acestora în cadrul buclei neavând nici un efect asupra numărului de executări ale instrucțiunilor.

Exemplu:

```
    a=3
    FOR  i=1 TO a
        a=10
        ? i*2
    ENDFOR
```

In acest exemplu comenzile a=10 si ? i*2 vorfi executate de 3 ori (i va lua pe rand valorile 1, 2 si 3), chiar daca la prima executie a acestora o noua eveluare a lui <expN2> ar duce la o noua valoare finala a contorului (aceasta evaluare nu mai are loc).

In schimb modificarea valorii contoruluiîn interiorul buclei va inflyenta numarul de executii ale grupului de instructiuni,

testarea valorii contor facandu-se la fiecare noua executie a <instructiuni>.
Exemplu:

```
FOR i = 1 TO 10
    ?i
    i = 15
ENDFOR
```

Comanda ? i va fi executata o singura data (pentru i = 1), dupa care datorita valorii 15 a contorului, care depaseste valoarea finala 10, se va iesi din bucla continuandu-se cu prima instrucțiune de dupa ENDFOR.

Doua comenzi speciale pot fi folositeîn interiorul buclei FOR….. ENDFOR:

EXIT determina iesirea fortata din bucla ;i continuarea executiei programului cu prima comanda care urmeaza dupa ENDFOR, indiferent de valoarea variabilei contor;

LOOP care determina saltul peste urmatoarele instructiuni ale buclei (dintre LOOP si ENDFOR), incrementarea sau decrementarea contorului si trecerea la o noua executare a grupului de instructiuni, daca se respecta conditia de ramanereîn bucla.
Exeplme:

```
Suma=0
FOR i=1 To 10
    Suma=suma+1
    IF i=5
        EXIT
    ENDIF
ENDFOR
```

Urmatorul program este identic ca rezultat, cu cel precedent:

```
Suma=0
FOR i=0 TO 10
    IF i>5
        LOOP
    ENDIF
    Suma=suma+1
ENDFOR
```

Primul program din acest exemplu functioneaza astfel: se executa comanada suma=suma+1 de 5 ori, pentru i=1,2,3,4 si 5.La i=5 este respectata conditia comenzii IF si deci fa executata comanda EXIT caredetermina saltul la ultima instrucțiunea programului.

Cel de-al 2 lea program din exemplul are urmatoarea functionare: se executa comanada suma=suma+1 de 5 ori, pentru i=1,2,3,4 si 5, atata timp cat nu este respectata conditia comenzii IF. Candaceasta conditie devine adevarata, pentru i=6,7,8,9 si 10, se va executa comanda LOOP care determina ignorarea comenzilor urmatoare din bucla, deci a comenzii suma=suma+1.Se va iesi din buclaîn mod normal, cand i=11.

Un tip special de bucla, foarte asemanatoare cu FOR ……….ENDFOR, dar specializatain lucrul pe o baza de date, este reprezentata de comanada SCAN…….ENDSCAN, cu sintaxa de forma:

SCAN [NOOPTIMIZE]
[<domeniu>] [FOR <exoL1>] [WHILE <expL2>]
<instructiuni>
[LOOP] [EXIT]
ENDSCAN

Aceasta comanda realizeaza parcurgerea bazei de date curente si executarea grupului de unstructiuni <instructiuni>,pentru fiecare inregistrare care aparatine domeniului specificat prin <domeniu>, FOR sau WHILE.

Clauzele LOOP si Exit au acelasi efect caîn cazul comenzi FOR……ENDFOR, iar clauza NOOPTIMIZE inhiba optimizarea RUSHMORE.

Exemplu: avem urmatoarele 4 echivalente:

1. SCAN 2. FOR I=1 TO RECCOUNT ( )
   <instructiuni>
   GOTO i
   ENDSCAN
   <instructiuni>

```
ENDFOR

3.  SCAN;
       NEXT 10 FOR MOD(RECNO( ),2)=0
       <instructiuni>
    ENDSCAN

4.  FOR i= RECNO ( )  TO  RECNO ( ) +9
       IF MOD(RECNO( ),2)=0
             <instructiuni>
       ENDIF
     ENDFOR
```

Cel de-al doilea tip de bucla, cu număr nedefinit de paşi, este implementat în FoxPro prin comanda DO WHILE…… ENDDO, avand sintaxa:

```
DO WHILE <expL>
       <instructiuni>
       [ LOOP ]
       { EXIT ]
ENDDO
```

Această comandă determină execuția repetată a grupului de instrctiuni <instructiuni>, atata timp cat valoarea expresiei logice <expl> este adevarata . Executia comenzii se va desfasura astfel:se evalueaza expresia <expl > si, daca acesta are valoarea .F. , executia comenzii se incheie. Daca valoarea acesteia este .T. se vor executa instructiunile din <instructiuni>.   La intalnirea lui ENDDO   se sare la linia continand DO WHILE si expresia <expL> este reevaluata. Acest ciclu se continua pana cand   o evaluare a expresiei logice <expL> va conduce la valoarea .F., cand executia comenzii DO WHILW …. ENDDO se incheie, programul continuand cu prima instrucțiune de dupa ENDDO.

Comenzile LOOP si EXIT au aceeasi semnificatie ca si la comanda FOR, prima determinand ignorarea restului de comenzi si reevaluarea lui <expL>, iar cea de-a doua determinand iesirea fortata din bucla, indiferent de valoarea expresiei logice <expL>.

Exemplu:

```
Suma=0
P=1
DO WHILE suma < 100
        suma= suma + p
        p= p+ 1
ENDDO
suma= suma + p
a= p- 1
?  ‘ S-au adunat primele ‘ , a , ‘ numere naturale’
? ‘ obtinandu-se suma de ‘, suma
```

acest program calculeaza suma a p numere naturale, p fiind numarul maxim de asemenea numere astfel incat suma lor sa nu depasesca 100.

Structura alternativă generalizată

Este o dezvoltare a structurii alternative pentru cazul când se continuă ramificarea algoritmului numai pe o ramură. Urmărind schema de mai jos, execuția unei structuri CASE are loc astfel: se evaluează prima condiție <c1> şi dacă este adevărată se execută secvența de comenzi <secv1> după care se părăseşte structura CASE.

Dacă condiția <c1> nu este adevărată se va testa condiția notată <c2>, ş.a.m.d. Se ajunge deci la testarea condiției <cn> atunci când nici una din condițiile precedente nu a avut valoarea adevărat.

Comanda DO CASE :

DO CASE
CASE <conditie1>
<secv1>
CASE <conditie2>
<secv2>
CASE <conditieN>
<secvn>
[OTHERWISE

<secv.m>]
ENDCASE

Comanda DO CASE este o comandă multi-linie.

Fiecare condiție, notată în formatul general prin construcția <cond1>... <condn>, este precedată de cuvântul cheie CASE. Dacă niciuna din condiții nu a fost adevărată, se va executa <secv.m> plasată după cuvântul OTHERWISE.

Depanarea programelor şi tratarea erorilor

În scrierea unui program , deseori apar erori datorate fie neatenției, fie nunei cunoasteri insuficientente a limbazului de programare, fie neluariîn seama a unor cazuri particulare ale evolutiei programului.

Doua mari clase de erori pot aparea la executia a unui program:

-erori de sintacsa, cand programul este incorect scris, ducand fie la intreruperea rularii, fie la executarea unor rutine pentru tratarea erorilor.

-erori de rulare, cand programul este corect scris, dar el nu functioneaza cum dorim, rezultatele obtinute nefiind cele asteptate de noi.

Primul tip de erori sunt mai usori de detectata si de corectata, pe cand cele din cea de-a doua grupa necesita, de obicei, mai mult pentru detectare si mai multa munca pentru corectare.

La rularea unui program, aparaitia uei erori de sintaxa determina, de obicei, intreruperea procesului de executie si afisarea unui mesaj de erore, indicand timpul erori aparaute si cerand utilizatorului luarea unei decizi, astfel:

-intreruperea programului, actionand declansatorul Cancel, care este si cel implicit;

-suspendarea executiei programului, obtinuta prin actionarea declansatorului Suspend;

-continuarea executiei, ignorand eroarea aparuta, prin alegerea declasatorului Ignore.

Detectarea erorilor de rulare presupune tehnici de lucru mai avansate, cum ar fi rularea pas cu pas, folosind punctele de intrerupere, vizualizarea continutului unor variabileîn paralel cu executarea programlor, etc.Vom prezentaîn continuare cateva tehnici folositeîn depanarea programelor realizate cu ajutorul adoua ferestre sistem ale FoxPro, fereastra Trace si fereastra Debug, acesibile din submeniul,în partea de jos a acestuia.

Rularea pas cu pas a unui program reprezinta o tehnica speciala folosita la depanarea programelor, constandîn exexcutarea unei singure instructiuni a programului la o comanda a utilizatorului.In intervalul de timp dintre executia a doua instructiuni consecutive, programatorul poate observa eventualele functionari incorecte ale programului, detectand astfel cauzele ce au dus la erorile de rulare respective.

Punctele de intrerupere reprezinta pozitii fixeîn cadrul programului de depanat, la care executia programului respectiv se opreste, utlizatorul avand satefel posibilitatea dea verifica starea de moment a variabilelor din program, modul cum decurge executia programului,în vederea detectari eventualelor surse de eroare.

Aceste doua tehnici prezentate mai sus sunt disponibile prin intermediul ferestrei Trace, care se deschide prin selectareaoptiuni Trace.

Pentru ca un program sa poat fie depanatîn fereastra Trace,acesta mai tebuie deschisîn acesta fereastra specificanduse programul de depanat. Dupa deschiderea programului se poate trece la executia lui, aceasta putanduse executaîn mai mute moduri:

-pas cu pas, cate o instrucțiune la fiecare comanda a utlizatorului;

-pana la un punct de intrerupere care va determina suspendarea programului;

-executarea continua, la viteza maxima;

-executarea continua cu viteza controlata, cand dupa executia fiecarei instructiuni se face o pauza dupa carea se continua rularea

Vom discuta mai rularea pas cu pas a unui program deschis în fereastra Trace. Pentru execuția fiecărei instrcțiuni a programului, utilizatorul va alege opțiunea Over sau Step, acestea determinănd execuția instrucțiunii curente, după care se aşteaptă o nouă comandă de executare de la utilizator

Diferența dintre cele două opțiuni este dată de modul de tratare a comenzilor de apel al unei rutine:

-Step determină executarea unei instrucțiuni a programului, iar în cazul când aceasta este un apel al unei rutine, se va trece în interiorul rutinei, pentru executarea instrucțiunilor acesteia;

-Over, de asemenea, execută o instrucțiune, dar când aceasta este un apel al unei rutine (program, procedură, funcție) aceasta este executată în întregime, într-un singur pas, fără a se trece la executarea pas cu pas a instrcțiunilor respectivei rutine.

Instrucțiunea curentă care va fi executată prin intermediul opțiunilor Step sau Over va apărea pe ecran supraluminată sau evidențiată prin intermediul unui semn, indicând această stare a liniei respective. În combinație cu instrucțiunea Step, se foloseşte şi opțiunea bară Out, care determină executarea continuă a restului de instrucțiuni din programul curent, urmând ca după revenirea în programul apelant, execuția să fie suspendată pe prima instrucțiune care urmează instrucțiunii de apel al rutinei. Acesta este echivalentă cu ieşirea din rutina curentă şi oprirea imediată în rutina apelantă.

O altă modalitate de rulare a uni program este cea prin puncte de întrerupere, adică, programul se va executa în mod continuu până la primul punc de întrerupere întâlnit. Pentru stabilirea punctelor de întrerupere, în care execuția programului va fi suspendată, se va deplasa cursorul pe linia respectivă, după care se va tasta Space sau Enter. O nouă acționare a tastei Space sau Enter, când cursorul se află pe o linie care conține un punct de întrerupere, va determina anularea punctului de întrerupere respectiv, linia revenind la starea inițială.

Ştergerea, anularea tuturor punctelor de întrerupere se poate face prin alegerea opțiunii Clear Breakpoints.

Un punct de întrerupere nu determină terminarea execției programului, ci numai suspendarea acestuia, până la o nouă comandă a utilizatorului cu privire la continuarea rulării. Opțiunea bară Resume determină continuarea execuției contunue a programului, până la un nuo punc de întrerupere, până la sfârşitul programului sau până la acționarea tastei Escape.

O modalitate specială de rulare a programului este obținută prin rularea opțiunii Throttle, constând în executarea continuă a

programului, instrucțiune cu instrucțiune făcându-se o pauză, stabilită de utilizator.

Oprirea execuției programului se poate face în orice moment, selectând opțiunea Cancel.

În paralel cu executarea unui program în fereastra Trace, se poate vizualiza conținutul unor variabile folosite în program, sau chiar rezultatul evaluării unor expresii folosind aceste variabile, prin intermediul ferestrei Debug.

Se recomandă ca fereastra Trace şi Debug să fie deschise şi vizibile simultan pa ecran,urmărindu-se astfel efectul fiecărei instrucțiuni executate în fereastra Trace asupra variabilelor din fereastra Debug.

Ferestra Debug este formată din două părți, astfel: în partea dreaptă se introdun variabilele şi expresiile care se doresc a fi vizualizate în timpul executării porgramului, în partea stângă apărând valorile de moment ale variabilelor sau expresiilor respective. De asemenea, în fereastra Debug se pot specifica puncte de întrerupere a programului dependente de valorile variabilelor sau expresiilor din fereastra respectivă, spre deosebire de punctele de întrerupere stabilite în fereastra Trace, care reprezintă puncte fixe ale programului, independente de valorile variabilelor.

Pentru a poziționa un astfel de punct de întrerupere, în dreptul variabilei sau expresiei dorite, pe bara ce separă cele două părți ale ferestrei, se va amplasa un marcaj. Acest lucru se va realiya fie cu mouse-ul , poyiționând cursorul acestuia în locu dorit şi acționând butonul stâng, fie prin intermediul tastaturii, deplasându-se pa bara separatoare folosind tasta Tab, poyiționîndu-se în dreptul variabilei sau expresiei dorite, cu ajutorul săgeților direcționale şi acționând tasta Space, pentru marcare. Execuția programului se va suspenda ori de câte ori valoarea variabilei sau expresiei corespunyătoare unui astfel de punct de întrerupere se va modifica.

Exemplu: dorim să calculăm suma primelor yece numere naturale, pentru aceasta sciind următorul program:

```
CLEAR
SET TALK OFF
suma= 0
FOR i = 0  TO 10
        suma = suma+i
```

ENDFOR

? ” Suma primelor yece numere naturale este:” , suma

acest program funcționaeyă corect, în urma executării lui obținându-se pe ecran mesajul:

Suma primelor yece numere naturale este 55

Dar să presupunem că la introducerea programului în memoria calculatorului în locul semnului “ + “ din instrucțiunea suma = suma + 1, s-a tastat semnul “ * “.

Suma primelor yece numere naturale este 0

Pentru detectarea erorii vom deschide cele două ferestre, Trace şi Debug simultan pe ecran, după care în fereastra Debug, în partea dreaptă, vom introduce variabilele de viyualiyat, suma şi I, iar în fereastra Trace, care va fi fereastra activă, vom deschide programul de depanat, SUMA.PRG, după care se va trece la rularea pas cu pas a programului.

Pentru aceasta se va selecta opțiunea bară Step a meniului ferestrei Trace, prin acționarea tastei “ S “ când fereastra respectivă este activă. Acționați de şase ori consecutiv tasta “ S “ obsevând în fereastra Debug modificările celor două variabile suma şi i, după fiecare executare a unei instrucțiuni. După cea de a şasea acționare a testei “ S “, ultima instrucțiune executată a fost sums = suma + i , care trebuie să aibă ca reyultat valoarea 1 depusă în variabila sums. În schimb noi vom observa că suma va avea valoarea 0 după executarea acestei comenyi, indicțnd că programul nu funcționeayă corect. Se va examina mai atent linia respectivă, detectându-se astfel eroarea de scriere (* în loc de +).

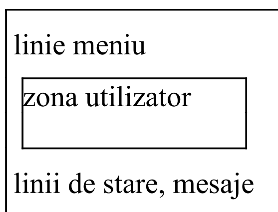
Pentru corectarea erorii se anulează executarea programului selectând opțiunea Cancel după care se reeditează programul într-o fereastră de editare.

Intrare / ieşire

Gestiunea ecranului

Ecranul are în mod uzual 24 linii şi 80 coloane dar, de fapt, zona utilizator este limitată la liniile [1, 21] pentru că linia 0 este linia de meniu principal, iar liniile [22, 24] sunt liniile de stare.

Putem avea acces la aceste linii prin folosirea unor comenzi-comutator.



linia zero poate fi activată/dezactivată cu SET SCOREBOARD ON/OFF

liniile 22,23,24 pot fi activate/dezactivate cu: SET STATUS ON/OFF

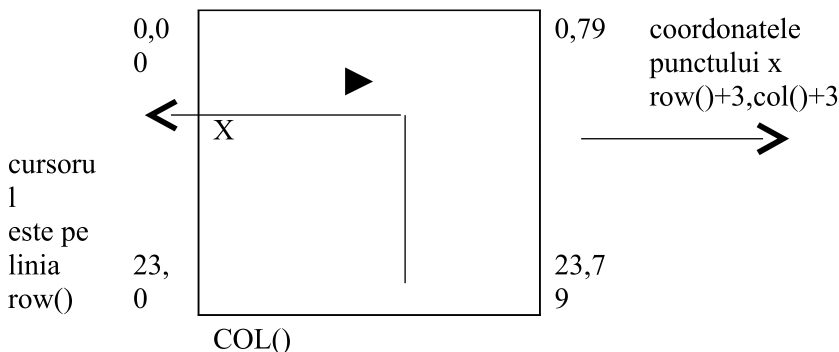
Observație: Numărul de linii şi de coloane depinde de modul de lucru al monitorului.

Există o comandă care poate fixa modul de lucru explicit:

SET DISPLAY MONO/COLOR/EGA25/EGA43/VGA25/VGA50

Pentru a indica o anumită poziție pe ecran, trebuiesc date linia şi coloana respectivă. Linia este un număr între 0 şi numărul maxim de linii al ecranului (standard 24), iar coloana este un număr între 0 şi numărul maxim de coloane al ecranului (standard 79).

****Observație*: În FoxPro există funcțiile SROW() care întoarce numărul maxim de linii al monitorului şi SCOL() care întoarce numărul maxim de coloane.



cursorul este pe coloana
col()

Unele comenzi şi funcţii de afişare pe ecran conţin explicit linia şi coloana unde se face afişarea respectivă, altele nu specifică poziţia de afişare ci folosesc poziţia cursorului de pe ecran.

Poziţia cursorului se poate afla prin funcţiile ROW() care întoarce numărul liniei şi COL() care întoarce numărul coloanei.

Gestiunea culorilor

Pentru că principalul dispozitiv periferic de ieşire este monitorul, desigur că un efect deosebit îl are afişarea datelor prin culori diferite în funcţie de importanţa lor.

Programatorul are la dispoziţie o paletă largă de culori, poate obţine efecte de intensitate sporită (bright) sau de pâlpâire (flash).

TABELE DE CULORI

nrc	culoare	cod
1	negru	N
2	albastru	B
3	verde	G
4	turcoaz	BG
5	invizibil	X
6	roşu	R
7	ciclamen	RB+
8	maro	GR
9	galben	GR+
10	alb	W

Culorile sunt indicate printr-un cod (abreviere).
Astfel, pentru a specifica culoarea ALB va trebui să folosim caracterul W, iar pentru roşu, caracterul R.

Putem obţine culori cu intensitate crescută punând ”+” după codul de culoare, iar efectul de ”flash” este obţinut prin caracterul ”*” după cod culoare.

De exemplu: gri închis N+, albastru intens B+, portocaliu R+, bleu BG+, violet deschis RB+.

Afişarea pe monitor a unui anumit caracter presupune specificarea a două culori: una pentru caracterul propriu-zis, numită culoarea cernelii (ink), iar cealaltă pentru fondul pe care se afişează caracterul, numită culoarea de fond (paper).

De exemplu: caracterul roşu pe alb se va indica prin R/W, galben pe negru GR+/N. Cele două culori separate prin caracterul ”/” (slash) se grupează într-o „pereche de culori”: descrisă în sintaxa comenzilor prin construcția <per-color> sau <ink>/<paper>.

Schimbarea stării color a monitorului se poate face prin comanda:

SET COLOR ON/OFF

Fixarea perechilor de culori pentru elementele principale ale ecranului se face prin:

SET COLOR TO [<std>][,<ext>][,<marg>][,<fond>]

Comanda fixează în clauza <std> perechea de culori pentru texte obişnuite; pentru text subliniat sau evidențiat (opțiunile selectate din meniuri, poziția cursorului la BROWSE etc.) este utilizată clauza <ext>, pentru margine clauza <marg>, iar pentru fond – clauza <fond>.

Exemplu:

set color to w+/b, gr+/n &&& afişaj standard litere albe pe fond albastru,
&&& extins litere galbene pe fond negru
set color to gr+, r, g, w &&& scriere cu litere galbene, selecția se observă
&&& cu roşu, bordura este verde

Revenirea la culorile setate în CONFIG.DB se face prin comanda:

SET COLOR TO

Fixarea explicită a perechilor de culori pentru zone ale ecranului se poate face prin

SET COLOR OF NORMAL/MESSAGES/TITLES/BOX/ HIGHLIGHT/ INFORMATION/ FIELD TO <per-color>

Comanda fixează culorile: pentru textul neselectat cu afişare obişnuită (NORMAL), mesaje
(MESSAGES), titluri, nume de câmpuri (TITLES), chenare (BOXES), informaţiile selectate (HIGHLIGHT), informaţiile sistem (INFORMATION), câmpuri (FIELDS).

Formatul de afişare şi citire

Formatul de afişare sau introducere a unei date reprezintă o deschiere simbolică a modului în care o dată este afişată pe dispozitivul de ieşire sau este citită de la tastatură.

Formatul se descrie prin două clauze: PICTURE, FUNCTION urmate de un şir de coduri. **Codurile FUNCTION** se referă la toate caracterele din format, iar **codurile din PICTURE** se referă la un singur caracter, cel de pe poziţia codului.
De exemplu: afişarea numărului 1234 depinde de şablonul deschis în clauza PICTURE, astfel:

picture ‘999’ .... > va determina trunchierea numărului
picture ’99, 999’ .... >determină introducerea separatorilor: 1, 234

Şablonul este util la introducerea de valori din exterior controlând tipul datei. De exemplu, dacă trebuie să citim numai cifre, atunci şablonul pus în comanda care realizează citirea va avea şablonul „99999” pentru ca, în caz de introducere a unui caracter nenumeric, să se avertizeze sonor eroarea.

Coduri de şablon pentru PICTURE

!	conversie în majuscule
#	afişează şirul definit de SET CURRENCY în locul unui zero

	nesemnificativ
*	afişează ”*” în locul unui zero nesemnificativ
9	permite cifre şi semnul pentru numere
,	separator (în numeric)
.	poziția punctului zecimal
A	permite doar litere
L	date logice
N	permite introducerea de litere şi cifre
X	permite introducerea oricărui caracter
Y	permite introducerea de valori logice {.Y./.y./.N./.n.}

Coduri de funcție pentru clauza FUNCTION

cod	Semnificație
!	converteşte toate literele în majuscule
^	afişează numărul în format ştiințific (mantisă şi exponent)
$	afişează simbolul monetar definit de SET CURRENCY
(	afişează numerele negative între paranteze
A	permite numai caractere alfabetice
B	aliniază textul la stânga unui câmp
C	afişează litera C (credit) după numerele pozitive
D	utilizarea formatului curent pentru dată calendaristică
E	utilizarea formatului european de dată calendaristică
I	centrează textul în cadrul unui câmp (numai SAY)
J	aliniază textul la dreapta în cadrul unui câmp numeric (numai SAY)
L	afişează zerourile nesemnificative într-un câmp numeric
M	defineşte opțiunile acceptabile pentru o variabilă GET
R	permite introducerea în şablon a unor caractere fără semnificație
S <n>	activează un mecanism de defilare orizontală într-o fereastră
T	elimină spațiile din față şi din spate în cadrul unui câmp
X	afişează DB(debit) după numerele negative
Z	înlocuieşte valorile nule cu spații

Comenzile ? şi ??

Comanda de afişare pe care am folosit-o până acum este ?/??. Ea poate avea şi clauze de specificare a formatului.
În format general comanda ?/?? este:

?/?? <exp> [PICTURE <sablon>][FUNCTION <functie>] [AT <coloana>]
[, <exp> [PICTURE <sablon>][FUNCTION <functie>] [AT <coloana>]..] [STYLE <tip-caracter>]

Se afişează valorile unei expresii sau a mai multora, conform clauzelor care le însoțesc: PICTURE <sablon> defineşte restricții şi formatul de afişare a fiecărui caracter din expresie: FUNCTION <funcții> specifică formatul global pentru expresie. Clauza AT <coloana> defineşte coloana unde se va afişa expresia. Clauza STYLE <tip-caracter> este o opțiune pentru imprimantă definind stilul de imprimare.

Ambele comenzi folosesc ca punc de referință în calculul coordonatelor de afişare, poziția curentă a cursorului. Comanda ?? începe afeşarea datelor exact de la poziția cursorului, pe când comanda ? începe afişarea datelor de la începutul primei linii care urmează celei pe care se află cursorul.

Dacă imprimanta nu suportă stilul specificat, atunci opțiunea se ignoră. Stilul este dat printr-o literă:
B: pentru caractere îngroşate;
I: pentru caractere italice;
U: pentru subliniere;
R: pentru indexare sus;
L: pentru indexare jos.

Exemple:

? (2 + 4) * 5

32

? “ Salutări din “
?? “ Bucureşti “

Salutări din Bucureşti

....? “ 2 + 3 = “ , 2 + 3

2 + 3 = 5

Observație:

Afişarea se face în mod implicit pe ecran, dar poate fi dirijată şi către imprimantă dacă există comanda de activare a imprimantei (comutator ON).

Comanda de activare/dezactivare a imprimantei este:

SET PRINTER ON /OFF

Comanda @ SAY

Este o altă comandă de afişare cu format:

@ <linie,coloană> SAY <exp> [PICTURE <sablon>] [FUNCTION <functie>][[OPEN] WINDOW <nume-fereastra>] [COLOR <per-color>]

Comanda afişează valoarea expresiei <exp> începând din punctul de coordonate <linie>, <coloană> conform şablonului dat în clauza PICTURE, şi/sau a funcției din FUNCTION. Clauza WINDOW permite afişarea într-o fereastră anterior definită a unui câmp memo. Clauza COLOR stabileşte atributele culoare sub forma <ink>/<paper>.

Exemplu:

```
alfa = “ Lucrez în FoxPro”
@ 10 ,10 SAY alfa
@ 12 ,10 SAY alfa SIZE 2 , 10
```

efectul este următorul:

prima comandă: **Lucrez în FoxPro**

a doua comandă: **Lucrez în**
FoxPro

Exemplu:

```
CLEAR
mesaj = “ la mulți ani ! “
@ 10 ,10 SAY mesaj PICTURE ' ! xxxxxxxxxxxxx '
```

La mulți ani !
@ 12 ,10 SAY mesaj FUNCTION ' ! '
LA MULȚI ANI !
@ 13 ,10 SAY mesaj PICTURE ' ! xx ! xxxxx ! xxxx '
La Mulți Ani !
@ 14 ,10 SAY 2 / 3 PICTURE ' 99 . 9999 '
0 . 6700
@ 15 ,10 SAY ' alternative ' FUNCTION ' ! x ! x ! x ! x ! x '
AlTeRnAtIv

Următoarele formate sunt echivalente :
PICTURE ' aaaaaa ' FUNCTION ' ! '
PICTURE '   ' FUNCTION ' a ! '
PICTURE ' @ ! aaaaaa '
PICTURE ' @ a !!!!!! '
Exemplu:
```
CLEAR
nume = '          '
prenume = '               '
data_n = {   /   /   }
data_căs = {   /   /   }
@ 10 ,10 GET nume
@ 10 ,10 GET prenume
@ 10 ,10 GET data_n
@ 10 ,10 GET data_căs  DISABLE
NOTE acest câmp este doar afişat dar nu poate fi accesat
datirită clauzei ;  DISABLE
READ
```
Exemplu:
```
CLEAR
STORE 0 TO  a , b
@ 10 ,10 GET a  PICTURE ' 9999 '
@ 10 ,10 GET b  PICTURE ' 9999 '  WHEN  a<> 0
NOTE al doilea număr va fi citit doar dacă primul este diferit
de zero
READ
```
Exemplu:
```
CLEAR
```

data_n = {   /   /   }
@ 10 ,10 GET data_n ;
MESSAGE '   Data naşterii în format ll / zz / aa   '
READ
? ' Născut la ' , data_n

Exemplu: pentru citirea unei valori numerice , situată între 0 şi 19 , se va folosi următoarea secvență:

CLEAR
a = 0
@ 10 ,10 GET a RANGE 0, 19
READ

iar pentru citirea unei date calendaristice, anterioară datei de 1 ianuarie 1970, se va folosi:

data = { }
@ 10 ,10 GET data   RANGE { 01 / 01/ 70 }
READ

Exemplu:

CLEAR
a = 0
b = SPACE (5)
c = '1 2 3 4 5 6 7 '
@ 10 ,10 GET a SIZE 1 , 4 && se va citi pe 4 caractere
@ 11 ,10 GET b SIZE 1 , 7 && se va citi pe 7 caractere
@ 12 ,10 GET c SIZE 1 , 5
NOTE citirea se va face într-o fereastră de 5 caractere, câmpul de ;
7 caractere defilând în această fereastră
READ
CLEAR
@ 10 ,10 GET a PICTURE ' 99999 ' SIZE 1 , 7 && se va citi pe 5 caractere
@ 11 ,10 GET b PICTURE ' xxxxxx ' SIZE 1 , 4
NOTE pe ecran se va edita în 4 caractere, câmpul GET de ;
7 caractere defilând în această fereastră
READ
CLEAR

Exemplu:

CLEAR

```
a = 0
b = 0
c = 0
@ 10 ,10 SAY  ' Introduceți numerele '
@ 11 ,15 GET a  PICTURE  '  99999 '
@ 11 ,10 GET b  PICTURE  '  99999 '  VALID a < b
NOTE  se  primeşte  numai  o  valoare  mai  mare  decât  cea anterioară
@ 13 ,15 GET c  PICTURE  '  99999 '  VALID c < c
READ
suma = a + b + c
 @ 14 ,15 SAY  suma     PICTURE  '  99999 '
```

Exemplu: se vor citi datele personale ale unui salariat, în vederea unor prelucrări ulterioare

```
SET TALK OFF
CLEAR
nume = SPACE ( 10 )
prenume = SPACE ( 15 )
data_n = {   /   /   }
strada = SPACE ( 15 )
nr = 0
localitate = SPACE ( 15 )
@ 5 ,10 SAY  ' Introduceți numele '
@ 5 ,40 GET nume  PICTURE  REPLICATE (' x ' , 10)
@ 6 ,10 SAY  ' Introduceți prenumele '
@ 6 ,40 GET prenume  PICTURE  REPLICATE (' x ' , 15)
@ 7 ,10 SAY  ' data naşterii '
@ 7 ,40 GET   data_n  FUNCTION ' d '  RANGE { 01 / 01 / 60 }
@ 8 ,10 SAY  ' Introduceți strada '
@ 8 ,40 GET strada  PICTURE  REPLICATE (' x ' , 15)
@ 9 ,10 SAY  ' numărul '
@ 9 ,40 GET   nr RANGE  1 , 999  PICTURE
@ 10 ,10 SAY  ' Introduceți localitate '
@ 10 ,40 GET localitate  PICTURE  REPLICATE (' x ' , 15) FUNCTION
READ
```

O comandă SAY şi o comandă GET pot fi grupate, formând o singură comandă :

```
        CLEAR
        @ 10 ,10 SAY  ' Introduceți un număr ' GET a  DEFAULT 0
PICTURE ' 99999 '
        READ
        @ 15 ,10 SAY  ' Ați introdus  numărul '
        ?? a
```

Exemplu: se citesc două numere strct pozitive, de maxim două cifre, astfel încât primul număr să fie mai mic decât al doilea, iar suma lor să fie egală cu 11. o primă variantă de rezolvare a acestei probleme este reprezentată de următorul porgram:

```
        SET TALK OFF
        CLEAR
        a = 0
        b = 0
        ? ' Introduceți 2 numere strict pozitive care însumate să dea
11'
        @ 10 ,10 SAY  ' Introduceți primul număr ' GET a
PICTURE  ' 99 '  VALID a > b
        @ 11 ,10 SAY  ' Introduceți al doilea număr '  GET b
PICTURE ' 99 ' ;
                VALID a < b  and a + b = 11 and b > 0
        READ
        ? a , ' < " , b , ' şi ' , a , ' + ', b , ' = ' , a + b
```

Acest program testează datele introduse de utilizator prin clauza VALID la nivelul comenzilor GET. Dacă, din greşeală, utilizatorul introduce 11 ca valoare a primului număr, la citirea celui de al doilea număr nu mai exeistă nici o posibilitate de a corecta eroare anterioară. Dacă se incearcă introducerea valorii 0 pentru cel de-al doilea număr (singura care corespunde condiției a + b = 11), clauza VALID a celei de-a doua comenzi GET se opune acestui lucru (trebuie ca b> 0 şi a < b). Nu se va permite nici măcar trecerea la câmpul GET anterior pentru a corecta eroarea şi a introduce o altă valoare pentru primul număr. Programul trebuie încheiat printr-o apăsare a tastei Esc , după care se poate reexecuta, pentru introducerea altor valori.

Problema aceasta îşi găseşte o rezolvare mai bună prin folosirea unei clauze VALID la nivelul comanzii READ, având, spre exemplu, următoarea variantă de program:

```
SET TALK OFF
CLEAR
a = 0
b = 0
? ' Introduceți 2 numere strict pozitive care însumate să dea 11'
@ 10 ,10 SAY ' Introduceți primul număr ' GET a PICTURE ' 99 ' VALID a > 0
@ 11 ,10 SAY ' Introduceți al doilea număr ' GET b PICTURE ' 99 ' VALID b > 0
READ VALID a < b and a + b = 11
? a , ' < " , b , ' şi ' , a , ' + ', b , ' = ' , a + b
```

În acest caz, dacă pentru primul număr se introduce valoarea 11, iar pentru cel de-al doilea număr valoarea 1, se va reveni în ultimul câmp GET. În acest moment se pot modifica ambele valori ale celor două variabile, testarea condițiilor a +b = 11 şi a < b făcându-se numai la ieşirea din comanda READ.

```
CLEAR
a = 0
b = 0
@ 10 ,10 SAY ' Introduceți primul număr ' GET a
@ 11 ,10 SAY ' Introduceți al doilea număr ' GET b
READ && citirea începe cu cel de-al doilea număr
? a + b
```

Exemplu: se vor citi patru numere, pozitive, aflate în progresie aritmetică, după care se va afişa suma acestora:

```
SET TALK OFF
CLEAR
a = 0
b = 0
c = 0
d = 0
@ 10 ,10 SAY ' Introduceți număr 1 ' GET a PICTURE ' 99 ' VALID a >= 0
```

@ 11 ,10 SAY ' Introduceți număr 2 ' GET b PICTURE ' 99 ' VALID b >= 0

@ 12 ,10 SAY ' Introduceți număr 3 ' GET c PICTURE ' 99 ' VALID c >= 0

@ 13 ,10 SAY ' Introduceți număr 4 ' GET d PICTURE ' 99 ' VALID d >= 0

READ OBJECT 2 TIMEOUT 5 NUMOUSE VALID a + d = b + c COLOR , R+/BG

? ' uma este :' , a + b +c + d

Citirea începe cu cel de-al doilea număr (OBJECT 2), mouse-ul nu poate fi folosit pentru selectarea câmpurilor (NOMOUSE), iar timpul maxim de gândire al utilizatorului este 5 secunde (TIMEOUT 5). Culorile folosite pentru câmpurile GET sunt : cerneală roşie pe fond albastru deschis. Condiția ca numerele să fie în progresie aritmetică este dată în clauza VALID a comenzii READ (a + b = c + d).

Observație: Afişarea expresiilor are loc implicit pe ecran dar poate fi direcționată şi către imprimantă, dacă aceasta este activată prin comanda SET PRINTER ON sau către un fişier de tip text. O comandă necesară este SET DEVICE.

SET DEVICE TO PRINTER/SCREEN/FILE <fis.txt>

Comanda direcționează efectul comenzilor de afişare către imprimantă, ecran, sau fişier de tip text.

Caracteristicile de culoare ale mediului FoxPro pot fi reținute în fişierul FOXUSER.DBF sub numele unui **set de culori**.

Un set de culori este o combinație de **24 scheme de culori**, fiecare schemă descriind un anumit obiect al mediului Fox: ferestre utilizator, fereastra BROWSE, meniuri şi submeniuri, zona de mesaje, etc.

O schemă de culori va avea un număr de **perechi de culori** (maxim 10) pentru a defini elementele obiectului respectiv. De exemplu, schema de culori pentru fereastra BROWSE va trebui să conțină perechile de culori pentru definirea cadrului, a interiorului, a numelui de câmp, a informațiilor selectate, a mesajelor.

Într-o schemă, perechile de culori de forma <ink>/<paper> sunt separate prin virgulă şi au o poziție semnificativă.

Setul de culori curent poate fi modificat indicându-se schema ce se modifică prin comanda:

SET COLOR OF SCHEME <nr1> TO <lista-per-cul>/SCHEMA <nr2>

unde <list-per-cul> specifică lista de perechi de culori care vor forma schema cu număr <nr1>.
Comanda de salvare pe disc în fişierul FOXUSER a schemei curente sub numele <nume-set>.

CREATE COLOR SET <nume-set>

Încărcarea unui set de culori din fişierul FOXUSER.DBF, salvat anterior, pentru a desemna setul curent se face prin comanda:

SET COLOR SET <nume-set>

Crearea unui set de culori precum şi salvarea pe disc se poate realiza şi interactiv, prin meniul WINDOW/COLOR.

Ordonarea unei baze de date

Multe din cererile de informații ale utilizatorului unei baze de date necesită parcurgerea ordonată a articolelor. Ordonarea datelor poate fi impusă de un timp scurt de regăsire a unui anumit articol: o căutare (LOCATE) secvențială se poate face într-un timp rezonabil pentru fişiere mici, dar pentru colecții mari de date tipul de răspuns la solicitate poate deveni inacceptabil de lung.
Operația de ordonare presupune *compararea articolelor între ele şi, în funcție de valorile cheilor de ordonare, accesul la un articol înaintea altuia.*
Criteriul de comparare între două articole poate fi **crescător** (asimilat cu un operator “>”) sau **descrescător** (asimilat cu operatorul “>”). Aplicat la două valori ale unei expresii, fiecare corespunzând uneia din înregistrările bazei de date, criteriul va da ca

rezultat adevărat (da, articolele se găsesc în relația cerută şi putem să ne folosim aşa cum se găsesc ele ca poziție!) sau fals (nu, articolele nu verifică relația, deci va trebui să facem în aşa fel încât să avem acces la cel de-al doilea articol înaintea primului !).
Operația de ordonare a unei baze de date se poate face prin două comenzi SORT şi INDEX, pe care le vom prezenta pe rând.

Sortarea datelor

SORT TO<fis.dbf>ON<câmp1>[/A][/D][/C]
[,<câmp2>[/A][/D][/C]
[DESCENDING/ASCENDING]
[<domeniu>][FOR <cond>][WHILE <cond>]

Comanda rearanjează fizic articolele bazei de date active, depunându-le într-o altă bază de date indicată în comandă prin clauza TO. Criteriul de ordonare poate fi unul sau mai multe câmpuri ale bazei de date.

Pentru fiecare cheie se specifică sensul ordonării. /A=ascending (crescător) /D=descending (descrescător) /C=se foloseşte pentru câmpuri de tip şir de caractere pentru a face compararea valorilor acestor câmpuri insensibilă la tipul literei (majusculă sau minusculă).
Clauzele ASCENDING/DESCENDING se folosesc atunci când toate cheile de sortare au acelaşi sens, fie crescător, fie descrescător.

Dacă odată cu clauzele locale de indicare a sensului asupra unei chei apar şi cele globale, primele au prioritate. Operația de rearanjare a articolelor se poate face nu pe întreaga bază de date (opțiune implicită) ci pe o parte a acesteia, parte fixată prin cauzele de selecție <domeniu>, FOR, WHILE.

Aranjarea articolelor atunci când sunt mai multe chei de sortare se face în modul următor: pentru aceleaşi valori ale primei chei se aranjează articolele după valorile celei de a doua chei. Dacă şi acum sunt valori egale, se trece la verificarea relației date de a treia cheie, ş.a.m.d.
Cheile de sortare nu pot fi câmpuri logice sau tip memo.

Exemplu: se ordonează baza de date Mijloacef.dbf, cheia de ordonare fiind codul mijlocului fix, iar ordinea crescătoare. Noua bayă de date se va numi Mfixe_s.dbf.

```
USE mijloacef
LIST
SORT TO mfixe_s ON cod / AC
USE mfixe_s
LIST
USE
```

Să se obțină din aceeaşi bază de date, lista tuturor mijloacelor fixe care nu sunt în folosință, în ordinea inversă a valorii, la valori egale ordinea fiind dată de data instalării, a punerii în funcțiune.

```
USE mijloacef
LIST
SORT TO mfixe_s ON valoare  / D, data_inst ASCENDING
        FOR stare = . F .
USE mfixe_s
LIST
USE
```

Sunt câteva particularități FoxPro legate de sortare.

1. Comanda SORT are o clauză în plus şi anume FIELDS <list-câmpuri> prin care se poate descrie structura bazei de date rezultate.
2. O altă particularitate este posibilitatea sortării tablourilor prin funcția ASORT:

ASORT (<tablou>[, <poz>[, <nr>[, <sens>]]])

Se sortează elementele tabloului în ordine crescătoare (dacă <sens>=0 sau lipseşte) sau în ordine descrescătoare (dacă <sens>≠0). Pentru a se putea sorta, toate elementele trebuie să fie de acelaşi tip.
Dacă masivul este unidimensional, atunci se vor sorta elementele acestuia începând cu elementul de pe poziția <poz>. Parametrul <nr> dă numărul elementelor care vor fi supuse operației de sortare. Dacă acesta lipseşte, se vor sorta toate elementele.

Indexarea bazelor de date

Indexarea presupune crearea unui fişier nou numit fişier-idex, asociat bazei de date, în care se memorează ordinea înregistrărilor din baza de date. Accesul la baza de date se face prin intermediul acestui fişier index.
Să luăm un exemplu: o bază de date în care avem încărcate 7 materiale, cu seria, cantitatea şi valoarea acestora. Indexarea acestei baze de date, după valoare, în ordinea crescătoare, presupune crearea fişierului MATERIAL.CDX în care se vor memora poziţiile înregistrărilor din baza de date, în ordinea dorită. Accesul la înregistrări se face prin intermediul fişierului index asociat .
Indexii pot fi depuşi:
1. În fişiere simple index, care reţin un singur index; ele trebuiesc deschise pentru a avea acces la acesta, fie în vederea parcurgerii ordonate a bazei de date asociate, fie pentru a reflecta şi în fişierul index modificările din baza de date. Dificultatea apare odată cu creşterea numărului de astfel de fişiere, care trebuie întreţinute şi actualizate.
2. În fişiere multi-index, care pot permite accesul, odată cu deschiderea multifişierului la toţi membrii lui.
Fişierele multiindex pot fi:

a. Asociate bazei de date (au acelaşi nume şi se deschid sau se închid odată cu baza de date, orice operaţie de actualizare asupra bazei reflectându-se automat şi asupra tuturor fişierelor index membre);

b. Izolate: au nume propriu diferite de al bazei de date, sunt create prin depunerea unui prim fişier index şi gestionate (deschise, actualizate, închise) prin comenzi explicite.

Fişiere index simple:

Crearea unui fişier index simplu se face prin comanda INDEX:

INDEX ON <exp> TO <fis.ndx> [UNIQUE]

Comanda permite crearea unui fişier index cu numele specificat în clauza TO având cheia de indexare dată în clauza ON <exp>. Clauza UNIQUE permite evitarea accesului cheii de indexare la articolele cu aceeaşi valoare. Acelaşi efect îl are şi comanda SET UNIQUE ON anterioară unei comenzi de creare (INDEX).

Exemplu: crearea unui fişier index simplu pentru baza de date mijloacef.dbf

```
USE mijloacef
INDEX ON cod TO mijloace_n
NOTE se crează fişierul index simplu mijloace_n.idx
LIST  && înregistrările sunt în ordinea câmpului cod
USE
```

Crearea unei etichete într-un fişier index compus nestructurat se face astfel:

```
USE mijloacef
INDEX ON valoare TAG val_n  OF mijloacef_d
NOTE se crează eticheta val_n în fişierul index compus nestructural mijloacef_d.cdx
LIST
USE
```

Adăugarea unei noi etichete la fişierul index anterior:

```
USE mijloacef
INDEX ON SUBSTR ( cod, 1, 4 ) + SUBSTR ( denumire  ,1 , 4 ) TAG codden  OF mijloacef_d;
        FOR data_inst  > { 01 / 01 / 90 }
NOTE se adaugă eticheta codden la fişierul index mijloacef_d se va prdona după cod şi denumire, doar;
        Primele patru caractere din aceste câmpuri fiind semnificative. Se selectează doar mijloacele; fixe ce s-au instalat după 1 ianuarie 1990
LIST
USE
```

Indexarea unei baye de date folosind un fişier index compus structural, în care se vor introduce patru etichete, corespunzătoare ordonării după câmpurile bazei de date:

```
USE mijloacef
INDEX ON  cod TAG  tcod
INDEX ON  valoare TAG   tvaloare
```

```
INDEX ON  amortizare TAG  tamortizare DESCENDING
INDEX ON  stare TAG   tstare
USE
```

Închiderea unui fişier index se face prin închiderea bazei de date asociată, (USE, CLOSE DATABASES), prin crearea unui alt index (INDEX ON) sau prin comenzi explicite de închidere:

SET INDEX TO **CLOSE INDEX**

Deschiderea unui fişier index se poate face şi odată cu deschiderea bazei de date căreia i-a fost asociat, prin indicarea numelui de index în comanda USE. Pentru că prin aceeaşi comandă se pot deschide şi alte fişiere index (simple sau multiple), vom nota lista acestora <lista-index> fără să precizăm tipul fişierului index.

USE <fis.dbf> INDEX <lista-index> ORDER <nume-index>

Exemplu: se deschide baza de date mijloacef şi odată cu ea o serie de fişiere index create în exemplele precedente.

```
CLOSE ALL
NOTE prima etichetă din mijloacef.cdx ca fi cea activă, adică ordonarea după cod
LIST
USE
USE mijloacef   INDEX   mijloacef_n.idx, mijloacef_d.cdx , mijloacef.cdx ORDER mijloacef_n
NOTE   se selectează ca activ primul fişier index simplu, cheia de indexare fiind câmpul cod
LIST
USE
USE mijloacef  INDEX  mijloacef  ORDER TAG  tvaloare
NOTE   se selectează eticheta  tvaloare din fişierul index compus structural mijloacef.cdx
LIST
USE
```

Deschiderea unui fişier index se poate face şi prin comanda explicită:

SET INDEX TO <lista-index> [ORDER <nume-index>]

Exemplu:

```
CLOSE ALL
USE mijloacef
SET INDEX TO  mijloacef ORDER TAG tcod ADDITIVE
LIST
USE
```

Chiar dacă prin comandă s-a deschis o listă de fişiere index, numai primul din listă este considerat principal. Clauza ORDER fixează la deschidere fişierul index principal.

Schimbarea ordinii de prioritate se face printr-o comandă explicită:

SET ORDER TO <nume-index>]

Exemplu:

```
CLOSE ALL
USE mijloacef  INDEX  mijloacef_n.idx, mijloacef_d.cdx , mijloacef.cdx
SET ORDER TO 2
LIST
SET ORDET TO codden OF mijloacef_d.
LIST
USE
```

Comanda permite indicarea indexului principal din lista de indexi deschişi.

Actualizarea unui fişier index nu se face automat dacă acesta nu a fost deschis în momentul actualizării bazei de date. O actualizare posibilă se face fie prin o nouă creare a fişierului fie prin comanda REINDEX:

REINDEX

Pentru o listă de fişiere index deschisă, comanda permite reactualizarea acestora în funcție de baza de date asociată.

Exemplu:

```
USE mijloacef  INDEX  mijloacef_n. mijloacef_d
```

REINDEX
USE

Funcţii standard relativ la indexare:

1) NDX ([[<nr-zona>][, <nr-ord>]]) întoarce numele fişierului index deschis într-o zonă specificată prin <nr-zona> (implicit zona ultim selectată) şi a cărui ordine în lista fişierelor active este <nr-ord>.

2) ORDER ([<nr-zona>]) întoarce numele fişierului index activ în zona dată prin numărul ei; implicit este zona curentă.

3) KEY ([[<nr-ord>][, <nr-zona>]]) întoarce expresia cheii de indexare a fişierului index identificat prin <nr-ord> în lista deschisă în zona de lucru indicată prin numărul ei, implicit zona curentă.

Căutarea rapidă şi poziționare în baza de date

Una din funcțiile importante ale unui SGBD este accesarea rapidă a bazei de date. Căutarea şi poziționarea pe un anumit articol se poate face prin mai multe comenzi şi funcții. Reamintim comanda LOCATE (caută secvențial un articol care verifică o condiție indicată în clauza FOR pe domeniul precizat), CONTINUE (reia căutarea în continuarea fişierului, poziționându-se pe următorul articol care verifică condiția), funcția LOOKUP (caută şi poziționează).
Alte posibilități de căutare rapidă sunt dependente de starea de indexare a bazei de date.

1. Comanda FIND permite căutarea într-o bază de date indexată a primului articol care are cheia de indexare egală cu expresia de căutare:

FIND <expC>/'<expC>'

Căutarea se opreşte la primul articol găsit, sau la sfârşitul fişierului.
<exp-şir> din comandă este încadrată obligatoriu de delimitatorii de şir, dacă expresia de căutare începe cu spații.

2. Comanda SEEK permite căutarea într-o bază de date indexată a primului articol care are aceeaşi valoare a cheii de indexare cu a expresiei din comandă:

SEEK <exp>

Comanda SEEK se execută întocmai ca şi comanda FIND, numai că <exp> de căutare nu mai este neapărat un şir ci poate fi orice variabilă, constantă sau expresie în general.
Căutarea se opreşte la primul articol din baza de date care are cheia de indexare egală cu valoarea expresiei – dacă o astfel de înregistrare există – sau la sfârşitul fişierului.
Exemplu:

```
CLOSE ALL
USE mijloacef INDEX mijloacef ORDER tcod
SEEK  "C1              "
? FOUND ( )
. T .
? EOF ( )
. F .
? RECNO  (  )
1
? RECNO  ( 0 )
0
DISPLAY
USE
```

Funcții de test asupra succesului sau insuccesului căutării sunt.
1) funcția FOUND() (întoarce .T. dacă articolul s-a găsit)
2) funcția EOF() (întoarce .T. dacă articolul nu s-a găsit)
3. Funcții de căutare:
SEEK (<exp>[,<nr-zona>/<alias>]) caută prima înregistrare pentru care cheia de indexare este egală cu <exp>. Dacă se găseşte funcția, întoarce valoarea .T., iar indicatorul de înregistrări se va poziționa pe înregistrarea găsită. Dacă nu se găseşte articolul dorit, funcția va

întoarce valoarea .F., iar indicatorul de înregistrare se va afla după ultimul articol.

Funcţia SEEK înlocuieşte secvenţa: - .SEEK<exp>
- ?.FOUND()

Calcule statistice cu datele din bazele de date

Scopul principal pentru care sunt create bazele de date îl reprezintă obţinerea într-un timp cât mai scurt, a unor informaţii cu privire la datele conţinute în bazele de date. Aceste informaţii pot fi de natură diferită, mai detaliate, sau mai sintetizate, sub formă de liste, tabele, sau simple valori, informaţii statistice, totalizatoare.

Comanda COUNT numără înregistrările din domeniul specificat prin < domeniu >, FOR şi WHILE, depunând rezultatul în variabila < var >, care va fi creată în cazul când nu există anterior execuţiei comenzii.

COUNT [TO <var>] [<domeniu>][FOR <cond>][WHILE <cond>]

Exemplu: numărarea mijloacelor fixe a căror valoare depăşeşte 10000000 se face prin următoarea secvenţă de comenzi:

```
CLOSE ALL
USE mijloacef
COUNT  FOR valoare > 10000000 TO nrmijloacef
? " În vaza de date avem " , nrmijloacef, " mijloace fixe "
? " cu  valoare mai mare decât 10000000 "
```

În baza de date avem 3 mijloace fixe
Cu valoare mai mare decât 10000000

```
USE
```

Comanda numără articolele din baza de date activă şi, dacă este prezentă clauza TO <var>, depune rezultatul în variabila specificată. Clauzele <domeniu>, FOR, WHILE limitează acţiunea comenzii.

Observaţie: Afişarea pe ecran a rezultatelor diferitelor comenzi dBASE (FoxPro) depinde de comutatorul SET TALK ON/OFF. Pentru valoarea OFF se inhibă afişarea rezultatelor.

Un alt tip de calcul ce se poate efectua asupra unei baze de date este sumarea valorii unor câmpuri numerice, din înregistrările selectate. Comanda folosită este SUM şi are sintaxa:

SUM [<lista-exp>] [TO <lista-var>/TO ARRAY <tablou>] [<domeniu>][FOR <cond>][WHILE <cond>]

Comanda permite însumarea valorilor fiecărei expresii în parte pentru toate articolelel bazei de date active din domeniul precizat în <domeniu>, care verifică condiția din FOR şi/sau condiția din clauza WHILE.

Clauza <lista-exp> conține expresii cu câmpuri ale bazei de date active.

Rezultatul poate fi afişat direct pe ecran (! atenție la starea comenzii SET TALK!), depus într-o listă de variabile sau într-un tablou (clauza TO ARRAY).

Implicit se vor însuma toate câmpurile numerice ale bazei de date şi rezultatele se vor afişa pe ecran.

Exemplu: având baza de date mijloacef.dbf, să se calculeze procentul valoric al amortizării relativ la valoarea totală a mijloacelor fixe:

```
USE mijloacef
SUM amortizare TO val_am
SUM VALOARE to val_tot
? " S-a amortizat " , val_am / val_tot * 100, " % "
? "   din valoarea totală a mijloacelor fixe "
```

S-a amortizat 42.33 %
din valoarea totală a mijloacelor fixe

```
USE
```

O comandă asemănătoare cu comanda SUM este AVERAGE

AVERAGE [<lista-exp>] [TO <lista-var>/TO ARRAY <tablou>][<domeniu>][FOR <cond>][WHILE <cond>]

Comanda permite calculul mediei aritmetice a valorilor expresiilor din <lista-exp>, pentru articolele din baza de date activă care se încadrează în domeniul precizat şi verifică condițiile din FOR şi WHILE.

Clauza <listă-expr> conține expresii având câmpuri ale bazei de date. Parametrul TO <listă-var> conține lista de variabile în care se depun valorile mediilor calculate. Rezultatele pot fi depuse şi într-un tablou (clauza TO ARRAY).

Dacă lipsesc toate clauzele, se va face media tuturor valorilor câmpurilor numerice, iar rezultatul se va afişa pe ecran (dacă SET TALK este ON).

Exemplu: să se calculeze valoarea medie a unui automobil folosind datele din baza de date mijloacef.

```
USE MIJLOACEF
AVERAGE VALOARE FOR UPPER ( SUBSTR ( cod, 1 , 1 )) = " A " TO medie
? " Valoarea medie a unui automobil: " , medie
```

Valoarea medie a unui automobil: 5432000

```
USE
```

Acelaşi lucru se poate realiza şi folosind comenzile SUM şi COUNT, cu ajutorul cărora se poate simula comanda AVERAGE:

```
USE MIJLOACEF
SUM VALOARE FOR UPPER ( SUBSTR ( cod, 1 ,1 )) = " A " TO val_tot
COUNT FOR UPPER ( SUBSTR ( cod, 1 ,1 )) = " A " TO nr_auto
medie = val_tot / nr_auto
? " Valoarea medie a unui automobil: " , medie
```

Valoarea medie a unui automobil: 5432000

```
USE
```

Pentru o serie de calcule financiare şi statistice asupra câmpurilor bazei de date se foloseşte comanda CALCULATE, cu următoarea sintaxă:

CALCULATE [<lista-exp>] [TO <lista-var>/TO ARRAY <tablou>][<domeniu>][FOR <cond>][WHILE <cond>]

Comanda poate calcula valorile mai multor expresii, pe care le depune într-o listă de variabile sau într-un tablou. Diferența față de comenzile anterioare constă în conținutul expresiilor din <listă-exp>.

În alcătuirea unei expresii pot intra următoarele funcții:
1. AVG(<expn>): permite calculul mediei aritmetice a expresiei <expn> ce poate conține câmpuri numerile ale bazei de date active;
2. CNT(): permite numărarea articolelor selectate din baza de date activă;
3. SUM(<expn>): permite însumarea valorilor expresiei <expn> ce conține câmpuri numerice a bazei de date;
4. MAX(<exp>): extrage cea mai mare valoare a expresiei <expn> calculată pentru fiecare articol selectat al bazei de date active;
5. MIN(<exp>): extrage cea mai mică valoare a expresiei calculată pentru fiecare articol selectat al bazei de date active.
Exemplu: să presupunem că avem o bază de date în care am stocat rezultatele unei experiențe, valori numerice. Vom lua spre exemplu următoarea serie de valori: 13, 47, 35, 9, 89, 123, 75, depozitate în câmpul număr al bazei de date NUMERE.DBF

```
USE  numere
CALCULATE  avg ( număr )  TO media
? " Media numerelor este : " , media
CALCULATE cnt ( )  TO nr_inreg
? " Numărul de valori este : " , nr_inreg
CALCULATE max ( număr ) , min ( număr ) , sum ( număr )
;
        TO maxim, minim, suma
? " Valoarea maximă este : " , maxim
? " Valoarea minimă este : " , minim
? " Suma numerelor : " , suma
CALCULATE npv ( 0.1, număr, 100 ) TO  val_p
? " Valoarea prezentă este : " , val_p
CALCULATE std (  număr ) TO  dev_std,
? " Deviația standard este : " , dev_std
? " Abaterea pătratică medie este " , ab_patr
USE
```

Un alt tip de rapoarte obținute dintr-o bază de date este reprezentat de listele totalizatoare. Comnda care realizează acest total este următoarea:

TOTAL ON <cheie> TO <fis.dbf> [FIELDS <lista-câmp>] [<domeniu>][FOR <cond>][WHILE <cond>]

TOTAL crează o nouă bază de date numită <fis.dbf> cu aceeaşi structură ca a bazei de date active. Baza de date activă este parcursă în întregime şi pentru fiecare grup de articole care au aceeaşi valoare a expresiei <cheie>, se adaugă în baza de date <fis.dbf> câte un articol cu cheia unică a grupului şi având în câmpurile numerice specificate în clauza FIELDS (sau toate câmpurile numerice dacă lipseşte clauza FIELDS) valoarea însumată a valorilor fiecărui articol din grup în câmpul respectiv.

Exemplu: pentru a afla suma de bani plătită de fiecare persoană din baza de date PLĂȚI .DBF, se foloseşte programul:

```
CLOSE ALL
USE plăți
INDEX  ON  plătitor  TAG plătitor  ORDER 1
TOTAL  TO  totplăți  ON  plătitor
SELECT 2
USE totplăți
LIST
SELECT 1
CLOSE ALL
```

Pentru cele trei înregistrări din baza de date, se va genera în baza de date totalizatoare TOTPLĂȚI.DBF înregistrarea care va conține suma totalizatoare.

Proceduri şi funcții definite de utilizator

Un program structurat este format din unități funcționale bine conturate, ierarhizate conform naturii intrinseci a problemei, obținut printr-un proces de rafinare treptată a prelucrărilor necesare rezolvării problemei.

Unul din principiile de bază din programarea structurată este programarea descendentă. Aceasta presupune descompunerea problemei complexe în subprobleme mai simple, descompunerea

continuând până la atingerea unui nivel de dificultate acceptabil. Fiecărei subprobleme îi va corespunde câte un modul-program cvasiindependent de celelalte module.
Avantajele programării modulare sunt următoarele:

a. structura problemei determină o definire precisă a funcțiilor fiecărui modul, fapt ce diminuează probabilitatea apariției erorilor de logică în rezolvarea problemei;
b. scrierea programului şi punerea lui la punct se face mai simplu şi mai rapid, deoarece modulele se pot realiza simultan, fiecare de către altă persoană;
c. modificarea sau extinderea programului se face mai uşor, prin modificarea sau scrierea unor module noi;
d. fiecare modul se poate scrie în limbajul de programare cel mai avantajos în raport cu funcția sa.

Modulele sau unitățile funcționale pot fi: subprograme, proceduri şi funcții. Vom examina fiecare tip de modul în parte:

Proceduri şi funcții

sunt unități funcționale de program tratate ca fişiere cu extensia .PRG. Se construiesc cu orice editor de texte. Apelarea editorului de programe se poate face prin comanda:

MODIFY COMMAND <fis.prg>

Lansarea în execuție a unui subprogram se face prin DO:

DO <fis.prg>

Comanda caută fişierul specificat, îl deschide şi, după execuția comenzilor, înaintea revenirii în programul apelant, îl închide.

O procedură reprezintă un grup de instrucțiuni ce primeşte de la programul apelant un grup de parametrii, realizează anumite prelcrări, după care se revine în programul apelant. O procedură definită de utilizator nu poate intra în alcătuirea unei expresii ca

operand. Trebuie să se facă distincţie între definiţia funcţiei sau procedurii şi apelul acesteia. La definirea unei funcţii sau proceduri se stabilesc prelucrările ce au loc în interiorul ei, parametrii care se primesc spre prelucrare şi rezultatele ce se vor transmite după prelucrare. La apelul unei funcţii sau proceduri apare doar numele care identifică respectiva funcţie sau procedură, însoţit eventual de lista parametrilor ce se vor transmite.

Variabilele definite în interiorul funcţiilor şi procedurilor sunt cunoscute doar în acest interval, în sensul că pentru programul ce apelează funcţia sau procedura, aceste variabile nu există.

O procedură începe cu comanda PROCEDURE şi conţine o serie de comenzi executate până la întâlnirea unei comenzi de sfârşit (RETURN, CANCEL, RETRY) sau până la o nouă comandă PROCEDURE.

Definirea unei proceduri se face prin comanda:

PROCEDURE <nume-procedură>

Apelul unei proceduri se face tot prin comanda DO prin care se lansează în execuţie programe sau subprograme:

DO <nume-procedură>

Revenirea în programul apelant se poate face prin comenzile RETURN, CANCEL, RETRY.
Comanda RETURN poate avea clauza TO MASTER care întoarce controlul în programul principal.

RETURN [TO MASTER]

Comanda CANCEL forţează renunţarea la toate unităţile program intermediare şi revenirea la prompterul sistemului dBASE (FoxPro):

CANCEL

Comanda RETRY permite revenirea chiar la instrucţiunea de apel a procedurii şi nu după aceasta, ca în cazul comenzii RETURN:

RETRY

O funcție reprezintă un grup de instrucțiuni independent, care primeşte un set de parametrii de la programul apelant şi îi returnează acestuia o valoare carezultat al prelucrărilor asupra parametrilor transmişi. O funcție definită de utilizator poate intra în componența unei expresii ca operand.

Definirea unei funcții se face prin comanda FUNCTION:

FUNCTION <nume-functie>

Apelul pentru execuția funcției se face prin numele acesteia în cadrul unei expresii. La execuție, în locul identificatorului se va introduce valoarea returnată de funcție ca rezultat al prelucrărilor sale.

Comunicarea rezultatului funcției se face prin comanda RETURN prezentă în corpul funcției:

RETURN <expr>

Transferul parametrilor la şi de la module de program

În mod uzual, comunicarea între unitățile funcționale ale unui program se face prin intermediul parametrilor. Parametrii permit ca la fiecare activare a unui modul, acesta să opteze cu entități care se cunosc doar în momentul apelului. Deoarece valorile acestea diferă de la un apel la altul, iar operațiile care li se aplică sunt aceleaşi, rezultă că instrucțiunile subprogramului vor trebui să se refere la entități exprimate simbolic, aşa numiții parametri formali. La activarea modulului se specifică parametrii actuali, sau efectivi.

Declararea parametrilor formali se face prin comanda PARAMETERS:

PARAMETERS <lista-var>

Această comandă , care trebuie să fie prima comandă a unui modul, defineşte lista de variabile locale care vor prelua parametrii transmişi de la programul apelant. Lista variabilelor locale trebuie să aibă întotdeauna mai multe elemente (sau cel mult egal) decât lista parametrilor transmişi, pentru ca fiecare parametru să aibă un corespondent în subprogram.

Exemplu:

```
SET TALK OFF
CLEAR
a = 14
b = 37
? a , " + " , b , " = " , suma ( a , b )
c = 12
d = 17
prod = 0
DO produs WHIT c , d , prod
? c , " + " , d , " = " , prod
FUNCTION suma
        PARAMETERS a 1, a 2
        RETURN a 1 + a 2
PROCEDURE produs
        PARAMETERS a 1, b 1, c 1
        c1 = a 1 + b 1
        RETURN
```

Parametrii formali sunt variabile locale unității funcționale.

Transmiterea parametrilor actuali se face prin comenzi de apel. Apelul unei proceduri se face prin comanda DO care va avea clauza WITH.

DO <identif> WITH <lista-exp>

Apelul unei funcții se face într-o expresie, parametrii fiind puşi între parantezele specifice funcției:

<nume-functie>(<lista-exp>)

Corespondența între parametri se face prin poziția în lista de parametrii din programul apelat şi din comanda de apel.

Exemplu:

```
SET TALK OFF
CLEAR
DO cadru
@ 12, 32 SAY mesaj ( )
READ
CLEAR

FUNCTION mesaj
        RETURN " Învățăm FoxPro "
PROCEDURE cadru
        @ 10 , 10, 14, 70  BOX
```

în acest exemplu DO cadru determină execuția procedurii cadru, definită după PROCEDURE cadru, iar mesaj ( ) este construcția de apelare a funcției mesaj, definită prin linia care urmează comenzii FUNCTION mesaj.

Să luăm următorul exemplu: calculul combinărilor, care se realizează pe baza formulei:

$$C_n^k = \frac{n!}{k! * (n-k)!}$$

Calculul factorialului n ! = 1 * 2 *………..*n se face cu următoarea secvență de instrucțiuni:

```
n=6
FACT = 1
FOR   i = 1  TO  n
        FACT = FACT * 1
ENDFOR
```

Pentru calculul combinărilor, această secvență se va repeta de trei ori, unde pe poziția lui n, vom avea pe rând n, k, n – k.

Programul pentru calculul combinărilor va fi:

```
n = 6
k = 3
 FACT1 = 1
FOR   i = 1  TO  6
          FACT = FACT * 1
ENDFOR
FACT2 = 1
FOR   i = 1  TO  k
          FACT = FACT * 1
ENDFOR
FACT3 = 1
FOR   i = 1  TO  n - k
          FACT = FACT * 1
ENDFOR
COMBIN = FACT1 / FACT 2 / FACT 3
```

Acest mod de scriere a unui program este ineficient pentru că acelaşi grup de instrucțiuni este scris de trei ori. În situații ca acestea, pentru scrierea programului se folosesc funcțiile şi procedurile definite de utilizator. Folosind pentru calculul factorialului o funcție, programul de calcul al combinărilor va căpăta următoarea formă:

```
n = 6
k = 3
COMBIN = FACT( n ) / FACT ( k ) / FACT ( n –
k )

FUNCTION FACT
PARAMETERS n
FACT = 1
FOR   i = 1  TO  n
          FACT = FACT * 1
ENDFOR
RETURN FACT
```

Primele trei linii reprezintă programul propriu-zis de calcul al combinărilor. În acesta se face de trei ori apel la funcția FACT ( ), care este definită în următoarele sapte linii de program.

Variabile globale şi locale

Variabilele definite într-o unitate program prin STORE, INPUT, etc. Există în memorie atâta timp cât programul este în execuție, fiind şterse automat la terminarea acestuia. Spunem că variabilele sunt locale sau private. Variabilele locale sunt recunoscute în subprogramele, procedurile sau funcțiile apelate din unitatea program care a definit variabilele, dar nu vor putea fi folosite în unitățile aflate pe acelaşi nivel sau superioare. În vederea comunicării cu unități funcționale superioare, variabilele se declară ca fiind globale sau publice.

Declararea variabilelor publice, cele care vor fi recunoscute în toate unitățile programului respectiv, se face prin comanda:

PUBLIC [ARRAY] <lista-var>

Variabilele simple se enumeră în <lista-var> fără să apară clauza ARRAY.

Exemplu:

```
SET TALK OFF
CLEAR
PRIVATE a
PUBLIC b
a = 1
b = 2
DO test
NOTE  aici se cunosc variabilele a , b , c dar nu se
cunoaşte variabila c
? " a = " , a
? " b = " , b
? " d= " , d

PROCEDURE  test
    PRIVATE  c
    PUBLIC  d
    c = 3
    d = 4
```

NOTE aici se cunosc toate variabilele: a , b , c , d

? " a = " , a
? " b = " , b
? " c = " , c
? " d = " , d

Statutul implicit al unei variabile este privat. Modificarea unei variabile private nu afectează valoarea variabilelor cu acelaşi nume din alte programe. Sunt situații însă când o unitate funcțională a fost concepută de altă persoană, sau în alt moment de timp şi ea foloseşte ca nume de variabile exact variabilele declarate public în alte unități program. Nu are rost să schimbăm numele variabilelor, peste tot unde apar: soluția este să le declarăm private.

Declararea variabilelor locale unei unități funcționale se face prin comanda:

PRIVATE <lista-var> / ALL LIKE / EXCEPT <salon>

Declararea variabilelor se poate face prin enumerarea lor în <lista-var>. Clauza ALL LIKE permite declararea privată a tuturor variabilelor care verifică un anumit <şablon>. Clauza ALL EXCEPT declară locale toate variabilele definite în programul respectiv cu excepția celor care verifică <şablon>.

Observații : în FoxPro sunt câteva particularități legate de proceduri:

1. Pentru a determina căutarea unei proceduri numai într-un anumit fişier, acesta se va include în clauzaîn a comenzii:

DO <nume-proc>în <fisier>

2. Folosirea unei variabile cu acelaşi nume în diferite părți ale unui program, este permisă dacă se declară regiunile programului cu comanda REGION:

#REGION <numar>

O regiune a programului ține până la o nouă declarare REGION sau până la sfârşitul programului. Comanda REGIONAL declară variabilele dintr-o listă ca fiind locale unei regiuni.

REGIONAL <lista-variabile>

Exemplu:

```
# REGION 1
REGIONAL a            && prima regiune
a = 1
? a
# REGION 2
REGIONAL a            && a doua regiune
a = 2
? a
```

în acest exemplu se folosesc două variabile cu acelaşi nume, a , acestea fiind definite ca regionale, în regiuni diferite. Deci comenzile a = 1 şi a = 2 se referă la variabile distincte.

Meniuri

În tendința generală de îmbunătățire a interfețelor cu utilizatorul ale aplicațiilor soft dezvoltate în ultima perioadă se înscrie şi înzestrarea acestora cu meniuri, dintre cele mai diverse şi mai performante. Alături de ferestre, meniurile dau Windows-ului, în general, şi FoxPro – ului, în particular, o putere deosebită, atât datorită eficienței şi comodității cu care se lucrează cu aceste elemente, cât şi datorită aspectului deosebit de plăcut pe care cele două elemente îl oferă programelor de aplicație şi mediului FoxPro.

Meniul reprezintă un element de interfață prin care se oferă utilizatorului posibilitatea executării unei anumite opțiuni, dintr-o mulțime finită de acțiuni posibile, afişată pe ecran total sau parțial, urmând ca, în funcție de selectarea făcută, să se declanşeze anumite acțiuni, corespunzătoare opțiunii alese.

Un meniu este format dintr-o “ bară a meniului” şi mai multe “submeniuri”. Bara meniului conține la rândul ei mai multe opțiuni, numite “opțiuni de bară”, fiecăreia dintre acestea putându-i-se ataşa un submeniu. Fiecare submeniu este format la rândul său din “opțiuni”.

Modul de lucru cu submeniurile definite de utilizator este următorul:

- mai întâi se defineşte meniul respectiv, prin următoarele etape:
- definirea barei meniului;
- definirea opțiunilor bară;
- definirea submeniurilor;
- definirea opțiunilor;
- definirea acțiunilor ce se execută la alegerea unei opțiuni sau a unei opțiuni bară a meniului;
- se activează meniul, urmând a se selecta de către utilizator opțiunea dorită;
- se efectuează diferite operații specifice meniului (afişare, ascundere, etc.);
- în final se elimină meniul din memorie, aceasta însemnând sfârşitul lucrului cu acest element.

Comenzi pentru lucrul cu meniuri

Definirea barei unui meniu se face prin intermediul comenzii DEFINE MENU:

DEFINE MENU < nume meniu >
[BAR [AT LINE < expN1 >]]
[în [WINDOW] < nume fereastră > | în SCREEN]
[KEY < etichetă tastă >]
[MARK < expC1 >]
[MESSAGE < expC2 >]
[NOMARGIN]
[COLOR <listă perechi culori>
|COLOR SCHEME <expN2>]

barei meniului I se atribuie un nume, < nume meniu >, urmând ca în continuare acest element să fie identificat prin numele atribuit.

Clauza BAR se foloseşte cu scopul de a prelua caracteristicile noii bare de meniu de la cea a meniului sistem al FoxPRO – ului. Următoarele aspecte sunt caracteristice meniului sistem:

- după alegerea unei opțiuni bara de meniu se dazactivează;
- bara de meniu va acoperi o singură linie a ecranului sau a ferestrei în care este plasată, de la un capăt la altul;
- poziția submeniurilor va fi stabilită de FoxPro, în mod automat;
- dacă bara va avea dimensiuni mai mari decât ecranul sau fereastra în care s-a introdus , se va folosi metoda defilării pentru accesarea opțiunilor bară.

Clauza AT LINE determina afişarea barei meniului pe linia cu numarul expN1 a ecranului sau a ferestrei respective.Clauzeleîn WINDOW siîn SCREEN sunt folosite pentru a specifica locul unde va fi plasata bara meniului:

-în fereastra nume fereastra,dacăîn comanda se include cauzaîn WINDOW;

-în fereastra curentă,dacă aceasta ezista si în camandă nu este prezentă nici una din cele două clauze;

-in ecranul FoxProW dacă se precizează clauzaîn SCREEN sau nu se specifică nici una din cele două clauze,iar ieşirea curentă este direcționată spre ecran.

Clauza MESSAGE determină afişarea şirului de caractere < expC > pe ultima linie a ecranului, în centrul acestuia sau în barea de stare de la partea inferioară a ferestrei, dacă aceasta există. Clauza este înfluențată de comanda SET MESSAGE. Opțiunile bară ameniului vor fi afişate una după alta, pe bara meniului, acestea fiind separate prin blancuri. Eliminarea blancurilor separatoare se face introducând clauza NOMARGIN în comanda DEFINE MENU respectivă.

Clauza COLOR şi COLOR SCHEME controlează culorile folosite pentru afişarea meniului. Implicit, afişarea meniului se va

face folosind culorile schemei 2 de culori, “ Meniuri definite de utilizator”, pentru folosirea altei cheme folosindu-se una dintre cele două clauze COLOE şi COLOR SCHEME.

După definirea unei bare a meniului trebuie să definim opțiunile bară care vor aparține barei respective, aceasta relizându-se cu comanda DEFINE PAD:

DEFINE PAD < opțiune bară > OF < nume meniu > PROMPT < expC1 >
[AT < linie >, < coloană >]
[BEFORE <opțiune bară2> | AFTER < opțiune bară3 >]
[KEY < etichetă tastă > [, < expc2 >]]
[MARK < expc3 >]
[SKIP [FOR < expL >]]
[MESSAGE < expc4 >]
[COLOR < listă perechi culori > | COLOR SCHEME < expn >]

Numele opțiunii bară va fi < opțiune bară >, iar bara de meniu de care aceasta va aparține va fi < nume meniu >. Textul care va fi afişat pe bara meniului va fi cel dat de clauza PROMPT, adică şirul de caractere < expC1 >. Stabilirea unei taste directe de alegere a opțiunii bară respective se realizează prin plasarea combinației “ \ < “ înaintea caracterului ce va desemna tasta directă, în textul clauzei PROMPT.

Exemplu: ……….. PROMPT “ \ < Stergere “ …..
Va defini un text a unei opțiuni bară, “Stergere “, în care tasta directă de alegere va fi “ S “.
Dacă în textul clauzei PROMPT caracterul selectat care reprezintă tasta directă nu este primul de acest tip, atunci ca tastă directă de alegere va fi aleasă prima apariție a caracterului respectiv din şir.

Exemplu: ………. PROMPT “Alb\ < astru ”…….
În acest caz vom avea textul “Albastru” în care “ A ” va reprezenta tasta directă de alegere (de fapt selectarea se poate face atât prin tastarea lui “ A “ cât şi tastarea lui “ a “).

Clauza AT determină poziția de pe ecran sau fereastra unde va fi afişată opțiunea bară respectivă: linia va fi desemnată prin < linie >, iar coloana prin <coloană>. În felul acesta se pot opține atât meniuri verticale şi orizontale, cât şi alte genuri de meniuri, neregulate.

În cazul în care clauza AT lipseşte, afişarea opțiunilor bară se va face începând de la linia 0, coloana 0, una după alta, pe direcția orizontală. Clauza AT nu se poate folosi pentru o bară a meniului definită cu clauza BAR (acestea sunt întotdeauna orizontale, iar poziția opțiunilor bară este controlată automat de către FoxPro).

Ordinea de apariție şi selectarea opțiunilor bară ale unui meniu este dată de ordinea definirii acestora prin comenzile DEFINE PAD corespunzătoare. Dacă după definirea unui număr de opțiuni bară se doreşte inserarea uneia noi între cele definite anterior, se foloseşte una dintre clauzele BEFORE sau AFTER.

Clauza BEFORE determină înserarea lui < opțiune bară1 > imediat înainte de < opțiune bară2 >, iar clauza AFTER determină înserarea lui < opțiune bara1 > imediat după < opțiune bară3 > .

Exemplu:

```
DEFINE MENU  test
DEFINE PAD opt1  OF test   PROMPT  ' Opt \ < 1 '
DEFINE PAD opt2  OF test    PROMPT  ' Opt \ < 2 '
DEFINE PAD opt3  OF test    PROMPT  ' Opt \ < 3 '
```

Acelaşi lucru se poate obține cu una din următoarele secvențe:

```
DEFINE MENU  test
DEFINE PAD opt1  OF test   PROMPT  ' Opt \ < 1 '
DEFINE PAD opt3  OF test    PROMPT  ' Opt \ < 3 '
DEFINE PAD opt2   OF test     PROMPT   ' Opt \ < 2 ' BEFORE opt3
```

Sau

```
DEFINE MENU  test
DEFINE PAD opt1  OF test   PROMPT  ' Opt \ < 1 '
DEFINE PAD opt3  OF test    PROMPT  ' Opt \ < 3 '
DEFINE PAD opt2   OF test     PROMPT   ' Opt \ < 2 ' AFTER opt1
```

test: Opt 1 Opt 2 Opt 3

clauza KEY se foloseşte pentru definirea unei căi directe de alegere a opțiunii bară respective. Calea directă de alegere reprezintă

o combinaţie de taste care acţionate la un moment dat, determină alegerea opţiunii respective. Combinaţia de taste va fi desemnată printr-o etichetă, < etichetă tastă >.

O opţiune bară care este prevăzută cu o cale directă de alegere va fi afişată având la dreapta ei eticheta respectivă. Dacă se doreşte inhibarea acestei afişări, sau afişarea unui alt text în locul etichetei, se va folosi şirul de caractere < expC2 > care poate conţine :

- textul de afişat în dreapta opţiunii bară, sau
- poate fi şirul vid, când se doreşte suprimarea afişării căii directe alături de opţiunea bară.

Exemplu:

DEFINE PAD opt1 OF test PROMPT ' Opt \ < 1 ' KEY Ctrl + H, “^H”

Opt 1 ^H **Opt 2** **Opt 3**

Pentru stabilirea condiţiilor de accesare a unei opţiuni bară se foloseşte clauza SKIP. Dacă această clauză se foloseşte fără FOR, optiunea bară respectivă nu va putea fi accesată, adică ea este dezactivată.

De asemenea dezactivarea unei opţiuni bară se poate face prin plasarea caracterului “ \ “ înaintea textului din clauza PROMPT.

Clauza MESSAGE determină afişarea textului din şirul < expC4 > cât timp opţiunea bară respectivă este selectată. Afişarea mesajului este controlată de comanda SET MESSAGE.

Exemplu: comenzile următoare sunt echivalente:

DEFINE PAD opt1 OF test PROMPT “ \ Alea “

DEFINE PAD opt1 OF test PROMPT “ \ Alea “ SKIP

Clauzele COLOR şi COLOR SCHEME specifică culorile folosite pentru afişarea opţiunii bară, implicit folosindu-se schema de culori 2.

Exemplu : vom defini bara de meniu de mai jos:

Modificare Adăugare ^N Ştergere
Listare Vizualizare Ieşire ^ K

în care opțiunile mai şterse sunt dezactivate, literele subliniate reprezintă taste directe de alegere, ^N , ^K reprezintă căi directe de alegere.

DEFINE MENU acțiune
DEFINE PAD mod OF acțiune PROMPT " \ < Modificare "
DEFINE PAD adaug OF acțiune PROMPT " \ < Adăugare " KEY Ctrl + N , ^N
DEFINE PAD sterg OF acțiune PROMPT " \ < Ştergere " SKIP BEFORE adaug
DEFINE PAD listare OF acțiune PROMPT " \ < Listare "
DEFINE PAD viz OF acțiune PROMPT " \ < Vizualizare "
DEFINE PAD ies OF acțiune PROMPT " \ < Ieşire " KEY Ctrl + X, "^ K " AFTER viz

Submeniurile se definesc prin comanda DEFINE POPUP :

DEFINE POPUP < nume submeniu >
[FROM < linie1 >, < coloană1 >]
[TO < linie2>, < coloană2>]
[în [WINDOW] <nume fereastră > | în SCREEN]
[FOOTER <exp C1 >] [TITLE < expC2 >]
[KEY < etichetă tastă >]
[MARGIN]
[MARK <expC3 >]
[MESSAGE <expC4 >]
[MOVER]
[MULTISELECT]
[PROMPT FIELD <expr> | PROMPT FILES [LIKE <mască>] | PROMPT STRUCTURE]
[RELATIVE]
[SCROLL]
]SHADOW]
[COLOR <listă perechi culori > | COLOR SCHEME <expN>]

Un submeniu reprezintă o listă de opţiuni care pot fi de următoarele tipuri:

-opţiuni definite de utilizator (prin comenzi DEFINE BAR);

-opţiuni calculate pe baza conţinutului unor înregistrări ale unei baze de date (PROMPT FIELD)

-câmpuri ale unei baze de date (PROMPT STRUCTURE);

-o listă de fişiere dintr-un anumit director, de pe un anumit disc (PROMPT FILES).

Numele submeniului va fi < nume submeniu >. Poziţia acestuia pe ecran sau într-o fereastră va fi dată de clauza FROM, în care < linie1 >, < coloană1 > indică poziţia colţului din stânga – sus al submeniului. Dacă această clauză lipseşte, submeniul va fi plasat în colţul din stânga – sus al ecranului FoxPro sau al ferestrei active, în poziţia 0,0

Clauza TO este folosită pentru a indica poziţia colţului din dreapta – jos al submeniului, prin coordonatele < linie2 >, < coloană2 >, acestea determinând şi mărimea submeniului pe ecran. În cazul când această clauză lipseşte, FoxPro va calcula automat dimensiunea submeniului.

Pentru ca submeniul să fie plasat într-o fereastră definită de utilizator, numele acesteia, < nume fereastră >, va fi inclus în clauzaîn WINDOW. Dacă această clauză lipseşte, submeniul va putea fi introdus într-o fereastră în cazul când ieşirea este direcţionată către acea fereastră (fereastra curentă) şi în comanda DEFINE POPUP nu s-a inclus clauzaîn SCREEN. Submeniul va fi plasat pe ecranul FoxPro dacă este prezentă clauzaîn SCREEN sau dacă nu s-a specificat nici clauzaîn WINDOW şi nici clauzaîn SCREEN şi ieşirea este direcţionată spre ecran.

Clauza TITLE se foloseşte pentru afişarea textului dat de şirul < expC2 >în centrul laturii superioare a chenarului submeniului, iar clauza FOOTER permite afişarea textului din şirul < expC! > în latura inferioară a chenarului submeniului. Clauza KEY se foloseşte în scopul de a specifica o cale directă pentru activarea submeniului, cale dată de eticheta < etichetă tastă >.

Cât timp submeniul este activat se poate afişa un text explicativ, un mesaj, pe ultima linie a ecranului sau a ferestrei respective. Mesajul este dat prin < expC4 > , expresie de tip şir de

caractere, inclusă în clauza MESSAGE. Clauza aceasta este controlată de comanda SET MESSAGE. Clauza MARGIN se include în comandă pentru ca opțiunile să fie separate de chenar printr-un spațiu suplimentar, la dreapta şi la stânga acestora.

MOVER este o clauză ce permite rearanjarea opțiunilor într-un submeniu, când acesta este activat. În prezența acestei clauze, opțiunile submeniului vor avea în dreapta lor caracterul "| " indicând posibilitatea de mutare a acestora. Clauza MOVER nu are efect în cazul submeniurilor create cu PROMPT.

O altă posibilitate a submeniului este cea a selecțiilor multiple, adică posibilitatea utilizatorului de a selecta mai mult de o opțiune. Fiecare dintre opțiunile selectate va fi marcată în stânga ei, cu un caracter de marcaj. Această facilitate este condiționată de prezența clauzei MULTISELECT în comanda DEFINE POPUP.

Modul de selectare a mai multor opțiuni depinde de comanda SET KEYCOMP. După realizarea unei multiselectări, testarea în program a selectării unei opțiuni a unui submeniu se realizează cu funcția MRKBAR(). Clauza MULTISELECT nu poate fi folosită în acelaşi timp cu clauza PROMPT.

Tipul de submeniu va fi determinat de clauza PROMPT. Dacă acestă clauză lipseşte, opțiunile submeniului urmează a fi definite cu comenzi de tipul DEFINE BAR.

În cazul clauzei PROMPT FIELD opțiunile submeniului vor fi preluate dintr-o bază de date deschisă într-una din zonele de lucru, pentru fiecare înregistrare câte o opțiune. La activarea submeniului se va selecta automat zona de lucru respectivă.

Expresia < expr > va determina modul de obținere a unei opțiuni din înregistrarea corespunzătoare. Această expresie poate conține câmpuri ale bazei de date deschisă în altă zonă de lucru, în care se poate aplica operatorul de concatenare. Numărul maxim de opțiuni ale unui submeniu creat dintr-o bază de date este nelimitat.

Exemplu:

```
CLEAR
USE agenda
DEFINE POPUP ag  FROM  10 ,10  TO 20 , 23 MARGIN ;
 PROMPT FIELD SUBSTR ( nume, 1 ,4 ) + " " + SUBSTR
( prenume, 1 , 4 )
ACTIVATE  POPUP  ag   && se activează meniul
```

CLEAR POPUPS && se şterge submeniul definit
USE

Cea de-a treia comandă a exemplului defineşte un submeniu numit ag, ale cărui opţiuni se obţin din primele patru caractere ale câmpurilor NUME şi PRENUME, concatenate, cu un blanc între ele.

Pentru ca opţiunile submeniului să reprezinte fişiere de pe un disc, dintr-un anumit dosar, se foloseşte clauza PROMPT FILES.

Exemplu: se va defini un submeniu în care se vor afişa toate bazele de date din rădăcina discului A:

DEFINE POPUP test FROM 10, 10 MARGIN PROMPT FILES A: \ *. DBF
ACTIVATE POPUP test
CLEAR POPUPS

Clauza SCROOL are ca efect afişarea unei bare de derulare verticale, pe latura dreapta a chenarului submeniului, când nu pot fi afişate simulta în submeniu toate opţiunile.

Pentru definirea opţiunilor uni submeniu, ce a fost definit anterior cu o comandă DEFINE POPUP, vom folosi comanda DEFINE BAR, cu sintaxa:

DEFINE BAR < expN1 > | < nume opţiune sistem > OF < nume submeniu > PROMPT < expC1 >
[BEFORE < expN2 > | AFTER < expN3 >]
[KEY <etichetă tasta > [, < expC2 >]]
[MARK < expC43>]
[MESSAGE <expC4 >]
[SKIP [FOR < expL >]]
[COLOR < listă perechi culori > | COLOR SCHEME < expN >]

Definirea opţiunilor într-un submeniu se face asemănător cu definirea opţiunilor bară, într-o bară a unui meniu , ca urmare comenzile DEFINE BAR şi DEFINE PAD sunt asemănătoare. Datorită acestui lucru, vom prezenta doar diferenţele dintre cele două comenzi.

Pentru fiecare opţiune a unui submeniu se va introduce câte o comandă DEFINE BAR. Identificarea opţiunilor într-un submeniu se face prin poziţia acesteia în cadrul submeniului.

În cadrul comenzii DEFINE BAR referirea la o opțiune se face prin < expN1 >, aceasta desemnând opțiune cu numărul de ordine < expN1 > a submeniului < nume submeniu >. În cadrul submeniului definit de utilizator pot intra ca opțiuni şi cele ale meniului sistem, această variantă fiind desemnată în comanda DEFINE BAR prin < nume opțiune sistem>.

Prin clauza PROMPT se specifică textul ce va fi afişat pe poziția opțiunii în submeniul respectiv. În şirul de caractere < expC1> ce desemnează textul respectiv, se pot introduce combinațiile “ \ < “, pentru desemnarea unei taste directe de alegere, sau “ \ “ pentru a indica o opțiune dezactivată.

Spre deosebire de comanda DEFINE PAD, şirul de caractere al clauzei PROMPT al comenzii DEFINE BAR, poate fi de forma “ _ “, indicând că pe poziția opțiunii respective se va introduce o linie de separare a opțiunilor, permițând astfel gruparea logică a acestora. Clauzele BEFORE şi AFTER sunt urmate de expresii numerice indicând “ înaintea “ căreia, respectiv “ după ce “ opțiune va fi introdusă noua opțiune a submeniului. Aceste clauze nu pot fi folosite decât în prezența clauzei RELATIVE în comanda de creare a submeniului.

În expresiile numerice ale acestor clauze pot fi incluse şi variabilele sistem _ MFIRST şi _MLAST, indicând primul şi, respectiv, ultimul articol dintr-un submeniu.

Exemplu: se va defini următorul submeniu:

Albastru
……………..
Verde
Roşu
Negru
Alb
Galben

```
DEFINE POPUP culori  FROM 6 , 10  TO 14, 21  MARGIN
MULTISELECT
DEFINE BAR 1  OF  culori PROMPT  “ Alb\<astru “
DEFINE BAR 2  OF  culori PROMPT  “ \<Verde “  SKIP
DEFINE BAR 3  OF  culori PROMPT  “ \<Roşu “
```

```
DEFINE BAR 4  OF  culori PROMPT  “ \_ “
DEFINE BAR 5  OF  culori PROMPT  “ \<Negru “
DEFINE BAR 6  OF  culori PROMPT  “ \<Alb “   SKIP
DEFINE BAR 7  OF  culori PROMPT  “ \<Galben“
ACTIVATE POPUP culori
CLEAR MENUS
```

Definirea unei bare de meniu sau a unui submeniu nu este suficientă pentru lucrul cu aceste elemente. Mai este necesară şi afişarea şi activarea lor, în acest mod bara de meniu sau submeniul apărând pe ecran şi cursorul trecând pe una din opțiunile elementului respectiv.

Afişarea şi activarea unei bare de meniu se face cu comanda ACTIVATE MENU.

ACTIVATE MENU < nume meniu >
[NOWAIT]
[PAD < nume opțiune bară >]

Comanda afişează şi activează bara < nume bară >, selectând inițial prima opțiune bară. pentru selectarea inițială a altei opțiuni bară se va folosi clauza PAD în care se va specifica opțiunea bară dorită.

Clauza NOWAIT determină continuarea execuției programului după afişarea şi activare barei de meniu (în mod normal programul se opreşte, aşteptând selectarea unei opțiuni bară). Bara de meniu va rămâne activă, utilizatorului dându-i-se posibilitatea selectării unei opțiuni bară ori de câte ori programul aşteaptă introducerea unor date de la tastatură.

Exemplu:

```
CLEAR
DEFINE MENU domeniu
DEFINE PAD fizica  OF domeniu PROMPT “ \<Fizica “
DEFINE PAD chimie  OF domeniu PROMPT “ \<Chimie “
DEFINE PAD literat  OF domeniu PROMPT “ \<Literatura “
DEFINE PAD sport  OF domeniu PROMPT “ \<Sport “
ACTIVATE MENU domeniu PAD sport  NOWAIT
```

```
@ 10, 10 SAY “ domeniu “ GET dom DEFAULT “ “
READ
CLEAR
DEACTIVATE MENU domeniu
```

Fizica Chimie Literatura
Sport

domeniu

Analog afişării şi activării barei de meniu se face şi afişarea şi activarea submeniului. În acest caz folosindu-se comanda ACTIVATE POPUP, cu sintaxa:

ACTIVATE POPUP < nume submeniu >
[AT < linie >, <coloană >]
[BAR < expN >]
[NOWAIT]
[REST]

Această comandă va afişa şi activa submeniul < nume meniu > , la coordonatele specificate în clauza AT, < linie > şi < coloană >.

Exemplu: bara de meniu domeniu, definită şi activată în exemplul anterior, o vom defini acum sub forma unui submeniu:

```
DEFINE POPUP domeniu FROM 10 , 10 MARGIN
DEFINE BAR 1 OF domeniu PROMPT “ \<Fizica “
DEFINE BAR 2 OF domeniu PROMPT “ \<Chimie “
DEFINE BAR 3 OF domeniu PROMPT “ \<Literatura “
DEFINE BAR 4 OF domeniu PROMPT “ \<Sport “
ACTIVATE POPUP domeniu BAR 4 NOWAIT AT 2, 5
@ 10, 10 SAY “ domeniu “ GET dom DEFAULT “ “
READ
CLEAR
DEACTIVATE POPUP domeniu
```

Fizică
Chimie
Literatură
Sport

domeniu 

Observăm analogiile dintre cele două exemple , care practic realizează acelaşi lucru, diferând doar formatul de afişare.

Afişarea pe ecran sau în fereastra activă , a barelor de meniu sau a submeniurilor fără activarea acestora, se realizează cu comenzile SHOW MENU ŞI SHOW POPUP cu următoarele sintaxe:

SHOW MENU < nume meniu1 > [, < nume meniu2 > ……] | ALL
[PAD < nume opțiune bară >]
[SAVE]

realizează afişarea barelor de meniu enumerate în lista ce urmează comenzii, sau a tuturor barelor de meniu, dacă în locul listei se foloseşte clauza ALL.

Pentru afişarea unei opțiuni bară ca fiind selectată, aceasta se specifică prin clauza PAD. Pentru reținerea imaginii bară de meniu pe ecran, fără activarea acesteia, se foloseşte clauza SAVE în comanda SHOW MENU. Această imagine va putea fi ştearsă prin comanda CLEAR.

Pentru afişarea submeniurilor vom folosi comanda SHOW POPUP.

SHOW POPUP < nume submeniu1 > [, < nume submeniu2 > ……] | ALL
[SAVE]

funcționarea acesteia fiind identică cu cea a comenzii anterioare.

Comanda HIDE MENU:

HIDE MENU < nume meniu1 > [, < nume meniu2 > ……] | ALL
[SAVE]

ascunde barelor de meniu enumerate în lista ce urmează comenzii, sau a tuturor barelor de meniu, dacă în locul listei se foloseşte clauza ALL, dar nu le elimină din memorie.

Comanda HIDE POPUP :

HIDE POPUP < nume submeniu1 > [, < nume submeniu2 > ……] | ALL
[SAVE]

ascunde submeniu enumerate în lista ce urmează comenzii, sau a tuturor barelor de meniu, dacă în locul listei se foloseşte clauza ALL

legătura dintre barele de meniu şi submeniuri este de o importanță deosebită pentru realizarea de meniuri complexe. O simplă bară de meniu sau un simplu meniu se poate folosi independent în program ca orice alt meniu, dar majoritatea aplicațiilor necesită meniuri mai complexe, care se obțin prin îmbinarea acestor elemente.

O primă modalitate de a lega la o bară de meniu un submeniu, sau o altă bară de meniu, este dată de comanda ON PAD:

ON PAD < nume opțiune bară > OF < nume meniu >
[ACTIVATE POPUP < nume submeniu >]
ACTIVATE MENU < nume meniu2 >]

Comanda va avea următorul efect: la alegerea opțiunii bară < nume opțiune bară > a barei de meniu < nume meniu1 > se va activa submeniul < nume submeniu > sau bara de meniu < nume meniu2 >, în funcție de clauza folosită în comandă.

Atenție: activarea submeniului sau a barei de meniu se va face la selectarea opțiunii bară respective) fără a fi nevoie de acționarea tastei Enter sau Space).

Pentru activarea unui submeniu se foloseşte clauza ACTIVATE POPUP urmată de numele submeniului, iar pentru

activarea unei alte bare de meniu se foloseşte clauza ACTIVATE MENU, după care se include numele barei respective. Dacă nu se foloseşte nici una dintre cele două clauze, alegerea opțiunii nu va mai determina activarea nici unuia dintre cele două elemente, comanda folosindu-se în acest caz la eliberarea opțiunii barei respective.

Exemplu: vom crea următorul meniu:

Calitate Culoare Preț
Stare
Negru
Satisfăcătoare
Alb
Bună
Albastru
Foarte bună
Verde
Roşu
Galben

Format dintr-o bară de meniu şi două submeniuri.

```
CLEAR
DEFINE MENU art
DEFINE PAD calit OF PROMPT “ \< Calitate “
DEFINE PAD cul OF PROMPT “ CU\<loare “
DEFINE PAD preț OF PROMPT “ \< Pret “
DEFINE PAD stare OF PROMPT “ \< Stare
DEFINE POPUP culori
DEFINE BAR 1 OF culori PROMPT “ \< Negru “
DEFINE BAR 2 OF culori PROMPT “ \< Alb “
DEFINE BAR 3 OF culori PROMPT “ AL\<bastru “
DEFINE BAR 4 OF culori PROMPT “ \< - “
DEFINE BAR 5 OF culori PROMPT “ \< Verde “
DEFINE BAR 6 OF culori PROMPT “ \< Roşu “
DEFINE BAR 7 OF culori PROMPT Galben “

DEFINE POPUP stări
```

```
DEFINE BAR 1 OF stări PROMPT “ \< Satisfăcătoare “
DEFINE BAR 2 OF stări PROMPT “ \< Bună “
DEFINE BAR 3 OF stări PROMPT “ \< Foarte bună “

ON PAD cul OF Art ACTIVATE POPUP culori
ON PAD stare OF Art ACTIVATE POPUP stări

ACTIVATE MENU art
DEACTIVATE MENU art
CLEAR POPUPS
CLEAR MENUS
```

Corespunzătoare comenzii ON PAD pentru opțiuni bară, avem o comandă pentru opțiunile unui submeniu, ON BAR, cu sintaxa:

ON BAR < expN > OF < nume submeniu1 >
[ACTIVATE POPUP < nume submeniu >] | ACTIVATE MENU < nume meniu >]

La selectarea opțiunii < expN > a submeniului < nume submeniu 1> se va activa submeniul < nume submeniu2 >, dacă se include clauza ACTIVATE POPUP, sau bara de meniu < nume meniu > , în cazul precizării clauzei ACTIVATE MENU. Dacă nici una din cele două clauze nu se include în comandă, opțiunea < expN > a submeniului < nume submeniu1 > va fi eliberată (nu va mai determina activarea unui element la alegerea sa).

Pot fi create în acest fel mai multe nivele de submeniuri, sau mai multe submeniuri care coordonează bare de meniu (mai rar).

Exemplu: la meniul creat anterior se adaugă două submeniuri de nivel doi, unul pentru opțiunea 3 (“ Albastru “) a submeniului culori şi altul pentru opțiunea 2 (“ Bună “) a submeniului stari.

```
CLEAR
DEFINE MENU art
DEFINE PAD calit OF PROMPT “ \< Calitate “
DEFINE PAD cul OF PROMPT “ CU\<loare “
DEFINE PAD preț OF PROMPT “ \< Pret “
```

```
DEFINE PAD stare OF PROMPT " \< Stare
DEFINE POPUP culori
DEFINE BAR 1 OF culori PROMPT " \< Negru
"
DEFINE BAR 2 OF culori PROMPT " \< Alb "
DEFINE BAR 3 OF culori PROMPT "
AL\<bastru "
DEFINE BAR 4 OF culori PROMPT " \< - "
DEFINE BAR 5 OF culori PROMPT " \< Verde
"
DEFINE BAR 6 OF culori PROMPT " \< Roşu "
DEFINE BAR 7 OF culori PROMPT Galben "

DEFINE POPUP stări
DEFINE BAR 1 OF stări PROMPT " \<
Satisfăcătoare "
DEFINE BAR 2 OF stări PROMPT " \< Bună "
DEFINE BAR 3 OF stări PROMPT " \< Foarte
bună "

ON PAD cul OF Art ACTIVATE POPUP culori
ON PAD stare OF Art ACTIVATE POPUP stări

DEFINE POPUP nuanța
DEFINE BAR 1 OF nuanța PROMPT " \<
Deschis "
DEFINE BAR 2 OF nuanța PROMPT " \<
Normal "
DEFINE BAR 3 OF nuanța PROMPT " \< Închis
"

DEFINE POPUP verif
DEFINE BAR 1 OF verif PROMPT " \<
Verificat "
DEFINE BAR 2 OF verif PROMPT " \<
Neverificat "

ON PAD cul OF Art ACTIVATE POPUP culori
```

ON PAD stare OF Art ACTIVATE POPUP stări
ON BAR 3 OF culori ACTIVATE POPUP nuanța
ON BAR 2 OF Stări ACTIVATE POPUP verif

ACTIVATE MENU art
DEACTIVATE MENU art

În acest exemplu avem următorul meniu:

Calitate Culoare Preț
Stare
Negru
Satisfăcătoare
Alb
Bună Verificat
Albastru Deschis
Foarte bună Neverificat
Verde Normal
Roșu Închis
Galben

Pentru ca la alegerea unei opțiuni bară, sau a unei opțiuni a unui submeniu, să se execute operații mai complexe vom folosi grupul de comenzi ON SELECTION, format din următoarele patru comenzi:

ON SELECTION MENU
ON SELECTION PAD
ON SELECTION POPUP
ON SELECTION BAR

Aceste comenzi determină executarea unei singure comenzi la alegerea unei opțiuni bară sau a unei opțiuni. Această comandă poate fi un apel de procedură sau o instrucțiune care să conțină apelul unei funcții definite de utilizator, în cadrul căreia se pot executa o mulțime de operații, în funcție de opțiunea aleasă.

Comanda ON SELECTION MENU are sintaxa:

ON SELECTION MENU < nume meniu > | ALL [< instrucțiune >]

< instrucțiune > va fi executată la alegerea oricărei opțiuni bară a barei de meniu < nume meniu > sau a oricărei bare de meniu definită

cînd se foloseşte clauza ALL. Comanda ON SELECTION MENU trebuie plasată între DEFINE MENU şi ACTIVATE MENU.

O utilizare asemănătoare o are comanda ON SELECTION PAD, cu sintaxa:

ON SELECTION PAD < nume opţiune bară > OF < nume meniu > [< instrucţiune >]

< instrucţiune > va fi executată numai la alegerea opţiunii bară < nume opţiune bară > a barei < nume meniu >.

Corespondentele celor două comenzi, pentru submeniuri, sunt:

ON SELECTION POPUP < nume submeniu > | ALL [< instrucţiune >]

ON SELECTION BAR < expN > OF < nume submeniu > [< instrucţiune >]

BIBLIOGRAFIE:

1.Dan marinescu, Mihai Trandafirescu
Manualul Începătorului, Ed. TEORA, Bucureşti, 1996
2.Ion Lungu ş.a.
Sistemul FOXPRO – Prezentare şi aplicaţii, Ed. ALL, Bucureşti, 1993.
3.Mariana Panţiru
Informatică Economică, Ed. PETRION, Bucureşti, 1996.
4.Lucian Vasiu, Sabin Ielceanu
FOXPRO- Programe, Editura Albastră, Cluj- Napoca, 1995.
5.Gabriel Dima, Mihai Dima
FOXPRO 2.6 sub WINDOWS, Ed. TEORA, Bucureşti, 1996.
6. Marin Fotache, Ioan Brava, ş.a.
VISUAL FOXPRO, Ed. POLIROM, Bucureşti, 2002.